

	Type	L #	Hit	Search T xt	DB	Time Stamp
1	BRS	L1	545	(tsirigotis).in. or (radia).in. or (chawla).in.	US- P PU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 11:47
2	BRS	L2	37	1 and cache	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 12:20
3	BRS	L3	0	("2003/0115420").URPN.	USPA T	2004/12/2 0 11:51
4	BRS	L4	0	("2004/0015725").URPN.	USPA T	2004/12/2 0 11:51
5	BRS	L5	0	("2004/0111443").URPN.	USPA T	2004/12/2 0 12:19
6	BRS	L6	27	2 and (web or object)	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 12:21

	<b>Typ</b>	<b>L #</b>	<b>Hit</b>	<b>Search T xt</b>	<b>DB</b>	<b>Tim Stamp</b>
<b>7</b>	<b>BRS</b>	<b>L7</b>	<b>73</b>	<b>(panagiotis and tsirigotis).in. or (sanjay and radia).in. or (rajeev and chawla).in.</b>	<b>US-PGPU B; USPA T; EPO; JPO; IBM_T DB</b>	<b>2004/12/20 12:21</b>
<b>8</b>	<b>BRS</b>	<b>L8</b>	<b>21</b>	<b>7 and 2</b>	<b>US-PGPU B; USPA T; EPO; JPO; IBM_T DB</b>	<b>2004/12/20 12:21</b>
<b>9</b>	<b>BRS</b>	<b>L9</b>	<b>16</b>	<b>8 and (web or object)</b>	<b>US-PGPU B; USPA T; EPO; JPO; IBM_T DB</b>	<b>2004/12/20 12:22</b>
<b>10</b>	<b>BRS</b>	<b>L10</b>	<b>8</b>	<b>8 and (web or object) same cache</b>	<b>US-PGPU B; USPA T; EPO; JP ; IBM_T DB</b>	<b>2004/12/20 12:22</b>

	Type	L #	Hit	Search Text	DB	Time Stamp
11	BRS	L11	0	("6754800").URPN.	USPA T	2004/12/2 0 12:25
12	BRS	L12	11	("6098096").URPN.	USPA T	2004/12/2 0 12:28
13	BRS	L13	5	cyclic adj5 buffer\$4 and 10	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 12:34
14	BRS	L14	1	cyclic adj5 buffer\$4 and primary and overflow and 10	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 12:36
15	BRS	L15	1	10/016,123	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:10

	Type	L #	Hit	Search Text	DB	Time Stamp
16	BRS	L16	1	buffer and 15	US-PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:16
17	BRS	L17	11	(overflow or primary) and 2	US-PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:17
18	BRS	L18	5	(overflow or primary) and 8	US-PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:20
19	BRS	L19	1177	(overflow or primary or cyclic) with (buffer\$ or cach\$4) with (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or new\$4)	US-PGPU B; USPA T; EPO; JP ; IBM_T DB	2004/12/2 0 13:25

	Type	L #	Hit	Search Text	DB	Time Stamp
20	BRS	L21	108	(overflow or primary or cyclic) with (buffer\$ or cach\$4) with (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or new\$4).ab.	US- P PU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:26
21	BRS	L20	118	(overflow or primary or cyclic) with (buffer\$ or cach\$4) with (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or new\$4).clm.	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 15:29
22	BRS	L22	0	15 and destination same available	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:37
23	BRS	L23	1	15 and destination same available	US- PGPU B; USPA T; EPO; JP ; IBM_T DB	2004/12/2 0 13:38

	Type	L #	Hits	Search Text	DBs	Time Stamp
24	BRS	L24	1	15 and available same space	US- P PU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:39
25	BRS	L26	1	15 and (available or space)	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:40
26	BRS	L27	1	15 and (updat\$4 or new or fresh) with buffer\$4	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 13:44
27	BRS	L28	120	(overflow or primary or cyclic) with (buffer\$ or cach\$4) with (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or n w\$4) with (free r availabl\$4 space)	US- PGPU B; USPA T; EPO; JP ; IBM_T DB	2004/12/2 0 15:30

	Type	L #	Hit	Search Text	DB	Time Stamp
28	BRS	L29	1	( verflow or primary or cyclic) with (buffer\$ or cach\$4) with (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or new\$4) with (free or availabl\$4 space).ab.	US- P PU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 15:30
29	BRS	L30	8	(overflow or primary or cyclic) with (buffer\$ or cach\$4) with (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or new\$4) with (free or availabl\$4 space).clm.	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 15:37
30	BRS	L31	1341	(overflow or primary or cyclic) same (buffer\$ or cach\$4) same (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or new\$4) same (free or availabl\$4 space)	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 15:37
31	BRS	L32	21	(overflow or primary or cyclic) same (buffer\$ or cach\$4) same (fresh or updat\$4 or up adj2 dat\$4 or refresh\$4 or new\$4) same (free or availabl\$4 spac ).clm.	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 15:59

	Type	L #	Hits	S arch Text	DB	Time Stamp
32	BRS	L33	3	"20020016911" or "20030093645" or "20020184441"	US- P PU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 16:00
33	BRS	L34	3	33 and (cache or buffer\$4) and (overflow\$4 or cyclic or free or available or updat\$4 or new or refresh\$4 or fresh or object)	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 16:19
34	BRS	L35	2	33 and (space or free)	US- PGPU B; USPA T; EPO; JPO; IBM_T DB	2004/12/2 0 16:20
35	BRS	L36	1	33 and (choose or chose or choice or select or pick) with (buffer\$4 or cach\$4)	US- PGPU B; USPA T; EPO; JP ; IBM_T DB	2004/12/2 0 16:21



US006754800B2

(12) **United States Patent**  
**Wong et al.**

(10) Patent No.: **US 6,754,800 B2**  
(45) Date of Patent: **Jun. 22, 2004**

(54) **METHODS AND APPARATUS FOR  
IMPLEMENTING HOST-BASED OBJECT  
STORAGE SCHEMES**

(75) Inventors: **Thomas K. Wong**, Pleasanton, CA  
(US); **Panagiotis Tsirigotis**, Mountain  
View, CA (US); **Sanjay R. Radia**,  
Fremont, CA (US); **Rajeev Chawla**,  
Union City, CA (US)

(73) Assignee: **Sun Microsystems, Inc.**, Santa Clara,  
CA (US)

(\*) Notice: Subject to any disclaimer, the term of this  
patent is extended or adjusted under 35  
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/987,249**

(22) Filed: **Nov. 14, 2001**

(65) **Prior Publication Data**

US 2003/0093645 A1 May 15, 2003

(51) Int. Cl.<sup>7</sup> ..... **G06E 12/00**

(52) U.S. Cl. .... **711/216; 711/118; 707/100**

(58) Field of Search ..... 711/216, 118,  
711/119; 709/203; 707/100

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,453,319 B1 \* 9/2002 Mattis et al. .... 707/100

\* cited by examiner

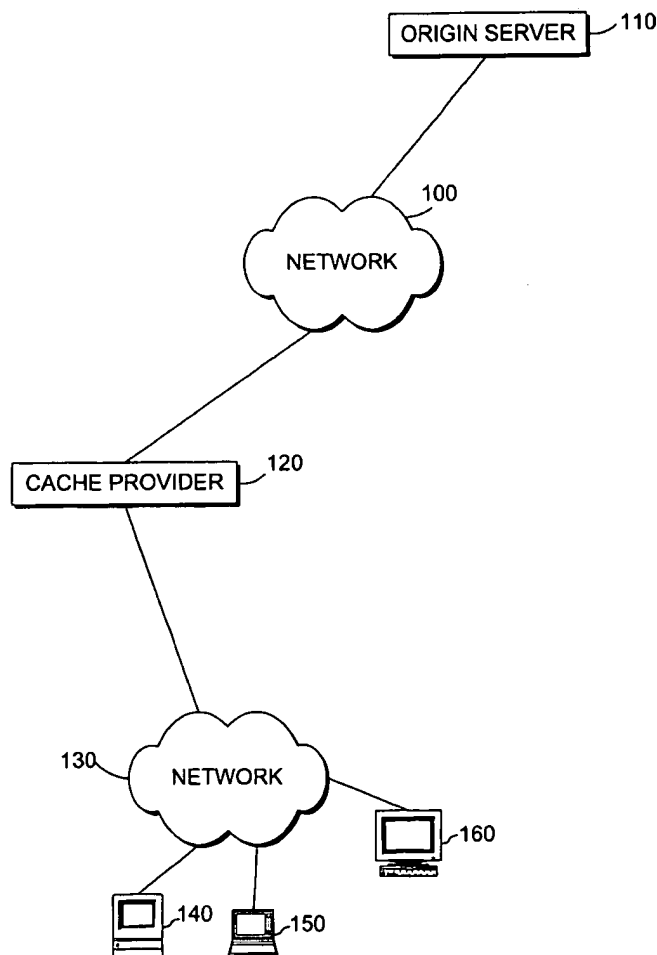
*Primary Examiner*—Kimberly McLean-Mayo

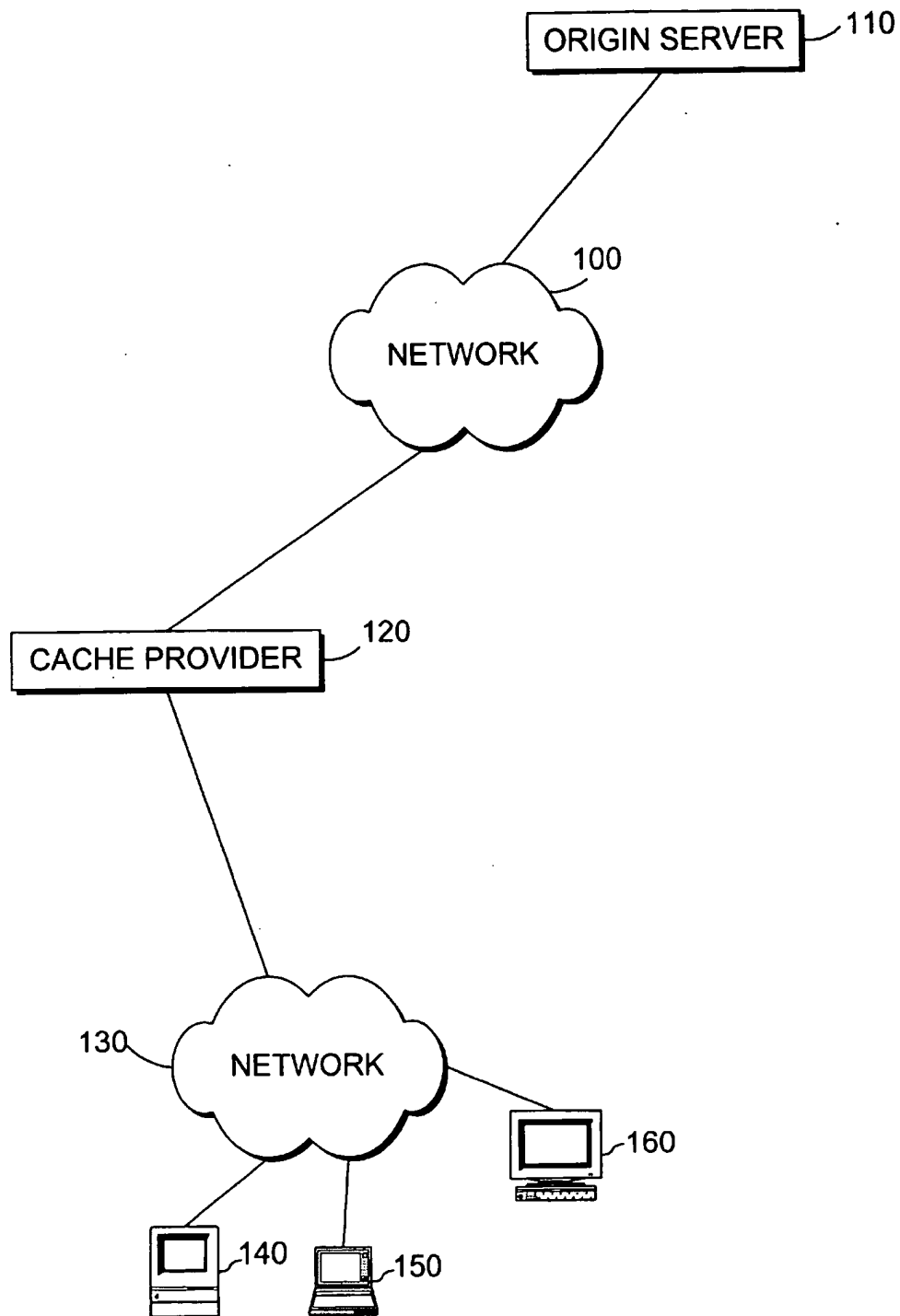
(74) *Attorney, Agent, or Firm*—Finnegan, Henderson,  
Farabow, Garrett & Dunner L.L.P.

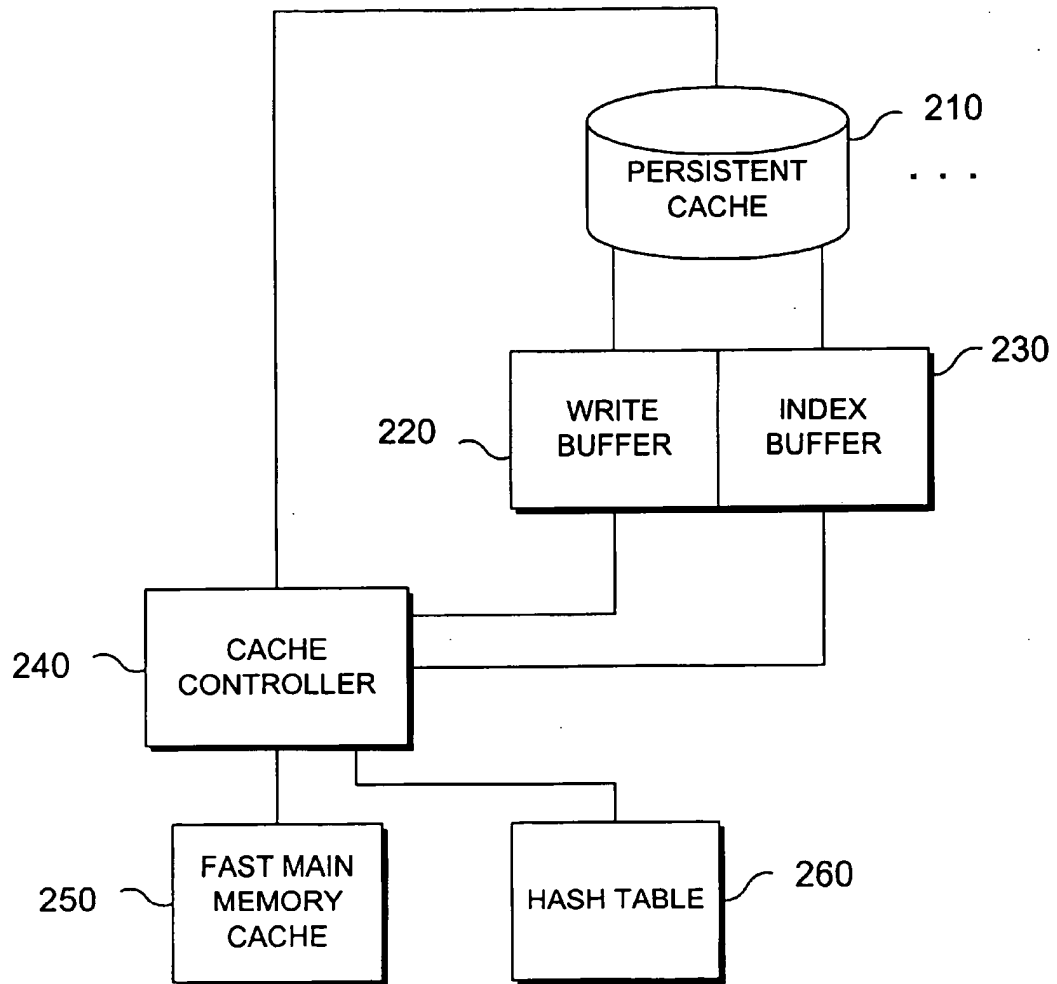
(57) **ABSTRACT**

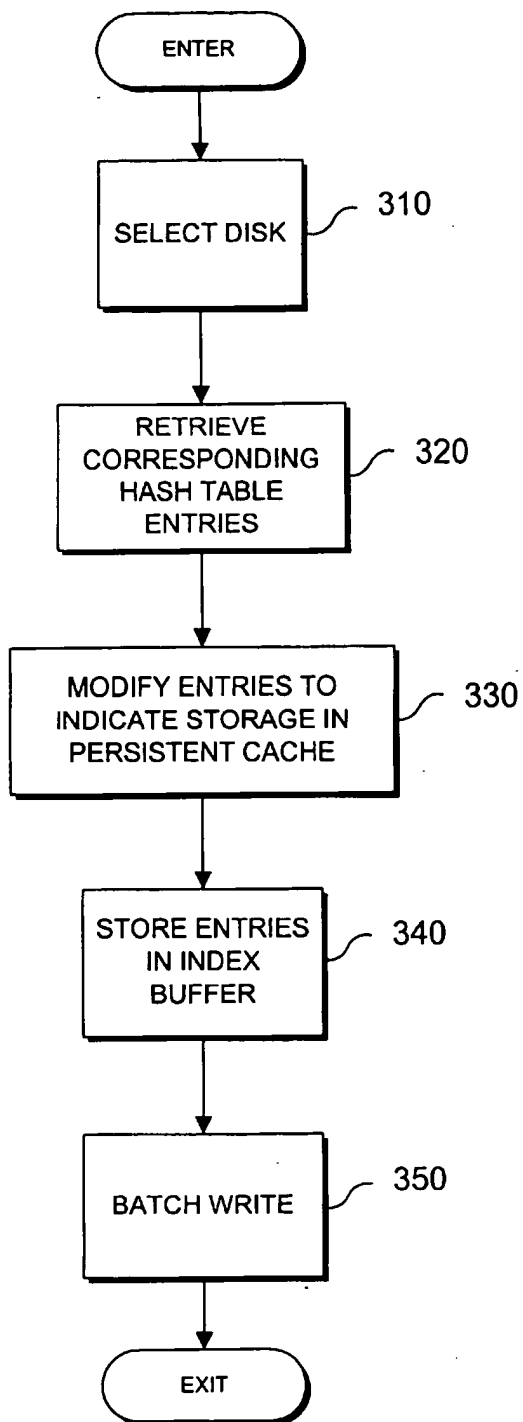
Systems and methods are disclosed in which a computer system having main memory and persistent memory is caused to perform a method for caching related objects. The computer system receives a plurality of objects from an origin server and computes a hash value based on source information about an object. Then the computer system stores the object based on the hash value with other related objects. Additionally, a computer system consistent with the present invention may retrieve related objects from the cache by performing a batch read of related objects.

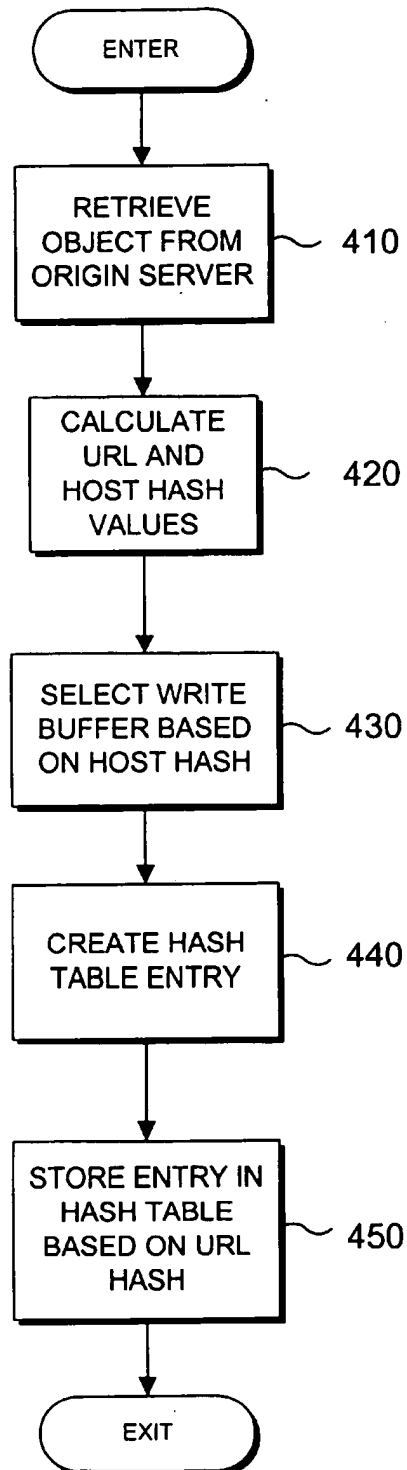
**26 Claims, 7 Drawing Sheets**



**Fig. 1**

**Fig. 2**

**Fig. 3**

**Fig. 4**

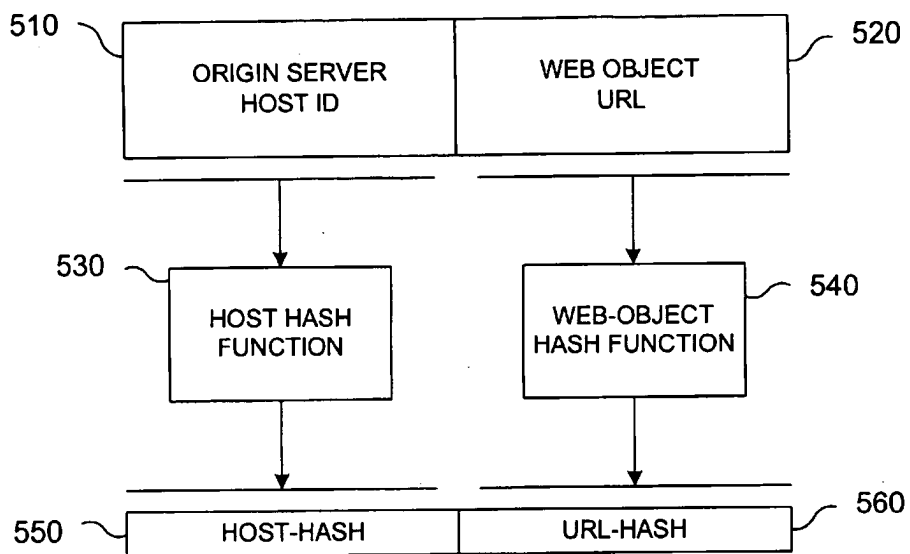


Fig. 5

HASH ENTRY

WEB OBJECT URL	SIZE	DISK ID	DISK LOC	(MEM ADDR)
<u>600</u>	<u>610</u>	<u>620</u>	<u>630</u>	<u>640</u>

Fig. 6

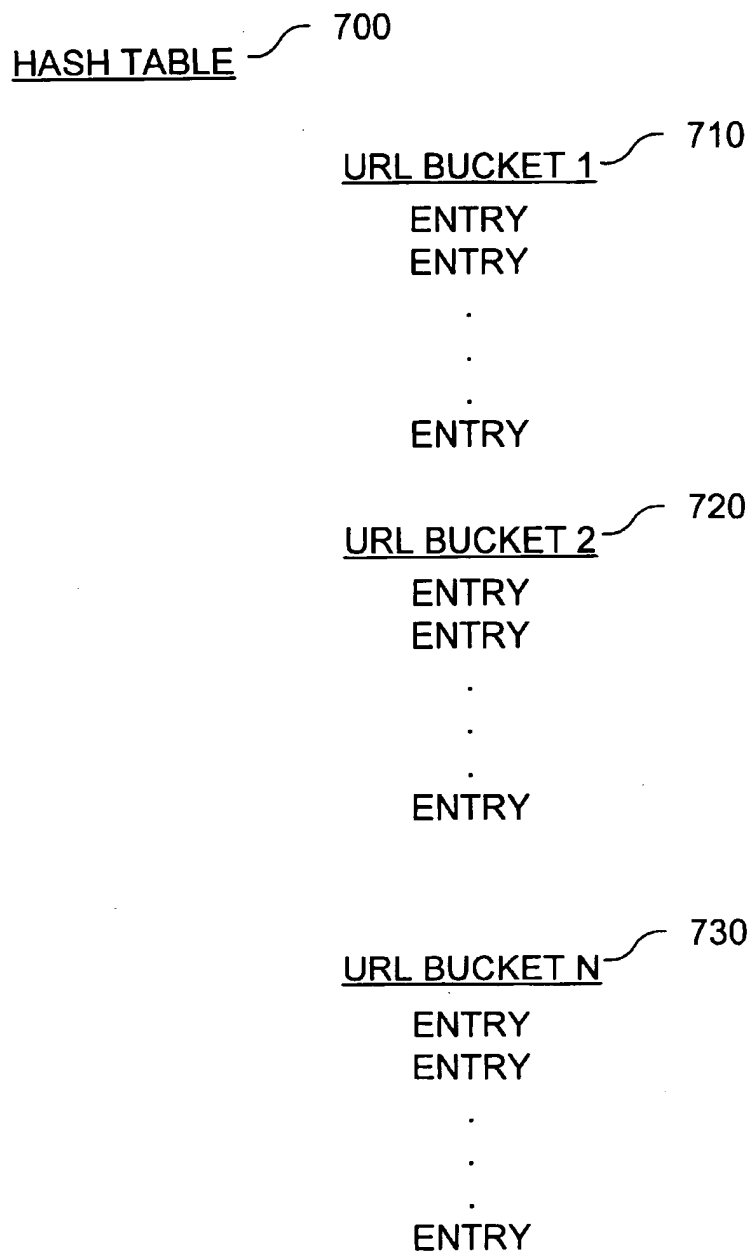


Fig. 7

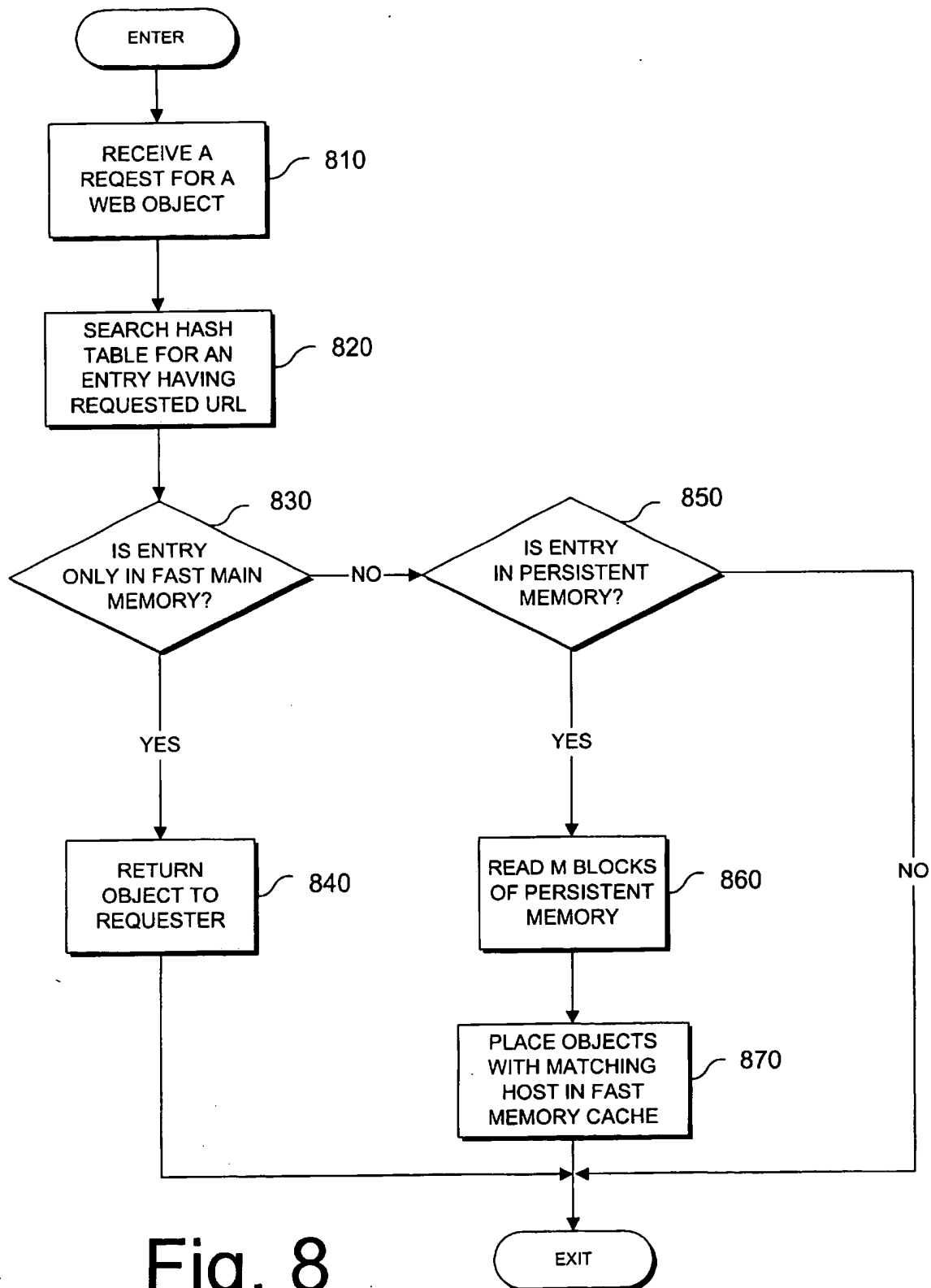


Fig. 8

1

## METHODS AND APPARATUS FOR IMPLEMENTING HOST-BASED OBJECT STORAGE SCHEMES

### DESCRIPTION OF THE INVENTION

#### 1. Field of the Invention

The present invention relates generally to caching objects and in particular to caching objects that are related to a common host computer system.

#### 2. Background of the Invention

The Internet is a world-wide collection of interconnected networks that is widely used to access information. A web server is an Internet-connected computer system that can provide information to Internet-connected client computers. Client computers use a computer program called a web browser to connect to and access information on web servers. Using a web browser, a user of an Internet-connected computer can access a huge number of different web servers. The vast collection of web servers available on the Internet is frequently referred to as the World Wide Web.

A web browser displays information received from a web server in units that have the appearance of pages, and because of their appearance, the pages downloaded from web servers and displayed by web browsers are frequently referred to as web pages. Web pages may contain references to elements including formatted text, images, and program code. The web pages and the referenced elements, such as image files and HTML documents, are sometimes called web objects. For example, when a web page contains a reference to a graphical image that should be displayed with the web page, the web page contains a reference to a graphic file. The graphic file and the file containing the information about the web page are called web objects. A general term for an information entity is "object."

As Internet access becomes more popular, less expensive and faster, and as the number of web clients increases, so does the number of connections made to particular origin servers. This increased number of connections can increase both network load and server load, sometimes causing parts of the Internet and particular origin servers to become so overloaded that they become inaccessible. Caching can be used to improve this situation.

A caching server may be used as an intermediary between a client and an origin server. Instead of a service request message being sent to an origin server, the message is routed to a proxy server. The proxy server handles the request by forwarding it to the origin server, receiving the requested information from the origin server, and transmitting the requested information to the client.

To expedite retrieval of requested information, a conventional proxy server often stores retrieved information in a local cache in the proxy server. When the proxy server receives a service request message from a client, it first determines whether the information requested by the service request message is already stored locally in the proxy server cache. If the requested information is stored in the proxy server cache, it is retrieved from the cache and returned to the client. If the information is not stored in the proxy server cache, the proxy server requests the information from the origin server. When the proxy server receives the requested information from the origin server, it sends the information to the client and also caches the information in the local proxy server cache. Caching can alternatively or additionally be done at other locations in a network. For example, a web browser can cache information locally at the client.

2

Various mechanisms for caching data are used in computer systems to reduce system loads by storing copies of frequently accessed information in places closer to where the information is likely to be needed. In response to a request from an information requester, a cache system determines whether the information is cached. If not, it is called a cache miss, and the cache system requests the information from the appropriate information provider, also saving a copy of the information in its local cache. If the information is already in the cache, it is called a cache hit. In either case, the information is then forwarded to the requester. By thus reducing loads and locating data closer to where it is needed, caching systems make data access more efficient.

Cache systems can improve the speed of information transfer between an information requester and an information provider. Cache systems generally include a high speed memory, such as, for example RAM, that stores information that was most recently requested from an information provider. A cache is often used in networks to expedite transferring information between a client and an origin server.

A protocol defines how the client and origin server communicate. The Internet, for example, uses protocols such as the Hypertext Transfer Protocol ("HTTP") to transfer web objects. In HTTP, a client sends a service request message to an origin server. For example, a service request message may include a request method to be performed on an object in the origin server. The object may be identified in the service request message by a name or an identifier of the object. In one embodiment, an object is represented as a web page and the identifier is a uniform resource locator ("URL"), which is a path to the object. The origin server responds to the service request message by performing the method in the request method. The request method may be a retrieve operation, which causes the origin server to retrieve the object identified by the URL, and transmit it to the requesting client.

Since the number of available documents in the web is huge, it is advantageous for a web caching system to use a large cache to increase the likelihood of a web hit. However, filling up a large cache with web objects may take a long time, potentially longer than acceptable, and it is therefore important to preserve the contents of a large cache during and after a system shutdown. Therefore, most web caching systems store cached web objects in a large but slow persistent storage, such as disk drives. Unfortunately, reading data from or writing data to a disk is slow, much slower than access to fast main memory. To reduce the delays associated with requests for web objects cached on disk drives, known caching systems use a fast-main-memory cache that is generally smaller than the persistent cache to store copies of web objects that are deemed most likely to be requested in the immediate future.

Typically, a web client accesses many pages from an origin server, one page at a time. Each page may refer to many other web objects including images and text. Typical web objects may be as small as 10 K bytes, or even smaller. To access a complete page that has not been previously cached by a web caching system, i.e. a cache miss, many separate requests must be made by a web browser. In the HTTP protocol, for example, this results in a separate request for each web object that is associated with a web page. Thus a cache miss for a web page usually results in one or more writes to disk for storing the retrieved objects referenced in the page. Similarly a cache hit for a page that is not in the fast main memory cache may result in one or more reads to the disk to retrieve each web object referenced

3

in the page. Thus, requesting a web object that is not in the fast main memory cache always results in at least one disk input or output ("I/O") operation, a minimum of one disk write for a cache miss and a minimum of one disk read for a cache hit. Since there are typically multiple web objects associated with a particular web page, this means that there will typically be multiple disk I/O operations associated with the caching or serving of a web page.

Disk access is significantly slower than fast main memory access. Typically, a disk can handle a small number of I/O operations per second. Since a cache hit or a cache miss for a web object usually results in at least one disk I/O, the cache throughput (number of web client requests that can be handled per second) of a web caching system is limited by the total number of I/O operations that can be supported by the disks used in the caching system. Thus, in some caching systems, disk access is the biggest bottleneck in maximizing cache throughput.

Strategically adding hardware to a caching system can increase the cache throughput. For example, if a larger fast-main-memory cache is used, more web objects may be stored in fast main memory, thereby reducing a probability of having to access a disk to retrieve a stored web object. Similarly, web objects may be distributed over multiple disks, thereby allowing multiple concurrent disk accesses and increasing the number of client requests that can be processed per second. While persistent storage devices are each relatively slow, they can operate in parallel, thereby increasing the total number of I/O operations per second.

Unfortunately, adding hardware to a caching system also increases the cost of the system. It is therefore advantageous to employ a design that improves caching throughput without depending on additional hardware. It is also desirable for such a design to scale-up in caching throughput with the use of additional hardware.

### SUMMARY OF THE INVENTION

Consistent with the present invention, systems, methods, and computer-readable media are involved in caching of related objects. A plurality of objects is received from at least one origin server. A hash value is computed corresponding to source identifying information for at least one object in the plurality of objects. In at least one data structure of a plurality of data structures, the object is stored based on the computed hash value, the object being stored with related objects having related hash values.

Additional benefits of the invention will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practice of the invention. The benefits of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention, as claimed.

### BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate exemplary embodiments of the invention and together with the description, serve to explain the principles of the invention. In the drawings,

FIG. 1 is a block diagram of a system in which the present invention may be practiced;

4

FIG. 2 is a block diagram of a cache system consistent with the present invention;

FIG. 3 is a flow diagram illustrating a method of batch writing objects consistent with the present invention;

FIG. 4 is a flow diagram illustrating a method, consistent with the present invention, of storing retrieved objects in a hash table;

FIG. 5 is a block diagram of a system, consistent with the present invention, for computing hash values;

FIG. 6 is a block diagram of a hash table consistent with the present invention;

FIG. 7 is a block diagram of a hash table, consistent with the present invention, with URL buckets; and

FIG. 8 is a flow diagram of a method, consistent with the present invention, for responding to a request from a requester.

### DETAILED DESCRIPTION OF THE EMBODIMENTS

Reference will now be made in detail to the present exemplary embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

A hash table is a data structure that utilizes a hash function to quickly locate an entry or a group of entries within a table of entries. Each entry or group of entries is identified by a hash value. A hash value is computed by applying a hash function to at least part of the information contained in a table entry. The entry or group of entries corresponding to a particular hash value is sometimes called a hash bucket. To add a new entry to a hash table, a hash function is applied to information within the new entry. If there is already an entry associated with the computed hash value corresponding to the new entry, an event called a hash collision occurs. There are several approaches that may be used in response to a hash collision. The choice of hash functions and schemes for handling hash collisions generally involves trade-offs between efficiencies of computation time versus an amount of storage space. It will be apparent to one of ordinary skill to choose a hash function and collision scheme that will work with the invention based on the anticipated size of the cache for a particular caching system.

Since objects that are referenced in a single page must be accessed at about the same time, it is desirable to store the related objects together, so that reading and writing of a batch of related objects may be done with a single read or write operation. The use of batch read and write operations increases cache throughput by reducing the number of relatively slow disk accesses per web page.

In one embodiment a file or a storage system is provided that caches web objects using identifiers called URLs, which may be represented as variable length character strings. For convenience, this embodiment may be referred to as the "urlfs," however, the invention applies to caching and storage of other kinds of objects, which may have different names or identifiers other than URLs. The urlfs consists of two caches, a fast main memory cache and a persistent cache that may consist of one or more disks. Web objects cached by urlfs may be stored in memory and in a disk or in a disk only. The fast main memory cache consists of a hash table, a set of write buffers and index buffers and an object cache that contains the newly retrieved or most likely referenced web objects.

The memory cache maintains a set of write buffers for web objects waiting to be written to a disk. When a new web

5

object is received from the origin server, the web object is first placed into the object cache. The URL-hash and host-hash for the new web object are computed and the host-hash is then used to select one of the write buffers to store the web object. A hash table entry for the web object is then created and the write buffer and the location within the write buffer where the web object is stored is entered into the hash table entry. The hash value of the URL of the web object is then used to locate a bucket in the hash table where the hash table entry should be stored.

In a urlls embodiment, each web object is stored in a variable size container called a cell, and each web object is contained entirely within a single cell. In this scheme, the size of a cell may be an integer multiple of the size of a persistent-storage data block, which may be, for example, the minimum storage allocation unit of a disk. Disks in the urlls have the same data block size. The attributes of a cell include the cell number, the size in number of data blocks, the URL-hash and the host-hash of the web object contained in the cell.

In the urlls, a web object is identified by a "URL." The URL of a web object also contains a variable size host identification that identifies the origin server of the web object. The URLs of web objects referenced in a page frequently contain the same host identification, but not always. For instance web objects may be duplicated on multiple servers, and a particular server may be referred to by multiple host names. Additionally, in many cases a web page may contain references to web objects that are stored on other origin servers.

Even though URLs and host identifiers are of variable size, a hash function can be used to generate fixed size hash values that are much smaller and that can facilitate efficient indexing of information within a hash table. One embodiment employs hash values from the URL of a web object and from the host identification of the web object to facilitate batch reading and writing of web objects referenced in the same page. The hash value of the URL of a web object is referred to as the URL-hash and the hash value of the host identifier of a web object is referred to as the host-hash.

Systems and methods consistent with the present invention may utilize cyclic buffers that use an index to map consecutively numbered logical blocks to physical buffer blocks of the cyclic buffer. The index to the cyclic buffer is implemented using the logical block numbers. When new information is added to the cyclic buffer, the next logical block number is assigned to the information. The new logical block number corresponds to a physical cyclic buffer block. The new logical block number and key associated with the information are then added to the index. The mapping of logical block numbers to physical block numbers may simply be a mathematical formula that defines the relationship between the logical block numbers and physical block numbers.

To retrieve information from the cyclic buffer, the index is first scanned to determine whether an index entry exists that has a key associated with the information. If such a key is present, the entry is accessed to determine the logical block number associated with the key. The logical block number may then be mapped to a physical block number, and the physical block number used to access data associated with a cyclic buffer. A physical block number corresponds to the actual hardware block to be read from a persistent storage device, whereas a logical block number corresponds to a software abstraction, which may be used to order physical blocks in various ways. For example, some

6

disk drives provide access to information in 512 byte blocks. A physical block number may correspond to a sequential ordering of these blocks, such that physical block one corresponds to the first block accessible by the hardware as designed and block two corresponds to the second block accessible by the hardware.

It may be convenient to use a block numbering system having logical block numbers that are different from the physical block numbers as implemented in the hardware of a persistent storage device. For example, it may be convenient to use a logical numbering system corresponding to a circular buffer, in which a particular logical block may not correspond to a physical block having the same number.

FIG. 1 is a block diagram of a system in which the present invention may be practiced. Clients 140, 150, and 160 are connected to network 130. Cache provider 120, connected between network 130 and network 100, handles service requests from clients 140, 150, and 160. In one embodiment, cache provider 120 may be a caching proxy web server. In another embodiment cache provider 120 may be a caching file server. In yet another embodiment, cache provider 120 may be a caching system that is implemented on a computer having client software, such as for example, in the case of a client side caching application. In response to a service request for information from a client, such as client 140, cache provider 120 either returns information from its local cache, or retrieves information from origin server 110, forwarding the information to client 140. If the information is not already in the local cache, the information is retrieved from origin server 110 and cache provider 120 then caches the information locally for later use. This is called a cache miss. Caching the information allows cache provider 120 to quickly provide the information to a client at a later time if the same information is requested again. When the information is found in the local cache and directly provided to the requesting client, it is called a cache hit.

FIG. 2 is a block diagram of a cache system consistent with the present invention. Cache provider 120 has a cache controller 240. Cache controller 240 is logically connected to fast main memory cache 250. Cache controller 240 is also logically connected to a hash table 260. In one embodiment, both fast main memory cache 250 and hash table 260 reside in fast main memory. Hash table 260 contains hash entries as further described in connection with FIG. 6. Cache controller 240 determines whether a web object is stored in persistent cache 210 or in fast main memory cache 250 by referring to hash table 260. For example, in response to a request for a particular web object, cache controller 240 looks up the web object in hash table 260. This process is further described in connection with FIG. 8. Cache controller 240 is also logically connected to write buffer 220 and index buffer 230.

Persistent cache 210 is cache memory, the contents of which remain valid even in situations where normal main memory would not retain its memory contents, such as when power is disconnected from the memory. Persistent cache 210 could be implemented, for example, using disk drives or other non-volatile memory devices. In one embodiment, persistent cache 210 comprises a plurality of disks that may be of different sizes. In the embodiment, each web object stored in persistent cache 210 has an index corresponding to each stored web object.

The persistent cache may include metadata. Metadata is data about data, and, in the urlls, it contains information about how data is stored and formatted within the persistent storage. The metadata may be stored at the same location on

7

each disk, and it stores information for accessing indexes corresponding to buffers within the persistent storage. In one embodiment, the metadata includes a timestamp, a start of index pointer, an index area size indicator, an index pointer, a start of data pointer, a data area size indicator and a data pointer.

To facilitate the storing of web objects in persistent cache 210, write buffer 220 is used to accumulate a group of web objects before they are written to persistent cache 210. This is because, in some types of persistent memory devices, such as disk drives, there is a fixed delay associated with writing to the device, regardless of the number of bytes being written. Therefore, efficiency is achieved by accumulating information in fast main memory before incurring the delay associated with writing to a relatively slow device. Index buffer 230 is used to locate a desired area within persistent cache 210 by associating an index value with a location in persistent cache 210 as further described in connection with FIG. 8.

FIG. 3 is a flow diagram illustrating a method of batch writing objects consistent with the present invention. When write buffer 210 becomes full or at other predetermined times, a disk is selected to store the contents of write buffer 210. First, the system determines which disk to use (box 310). There are several possible choices when selecting a disk within persistent cache 210 for storing contents of write buffer 220. For example, an idle disk or a least busy disk may be selected. One embodiment uses the host-hash to select a disk that has a disk number equal to the remainder of the host-hash divided by the number of disks in the storage system. Next, the system accesses hash table entries corresponding to the web objects contained in write buffer 220 (box 320). This is done by computing the URL-hash value corresponding to the web objects. Within the hash entry, the information about the web object is then modified to indicate the web object as being stored in persistent cache (box 330). One embodiment also records a disk identifier associated with the selected disk and updates the new cell number within the data buffer.

An index entry for a web object is then created using the contents from the hash table entry. The new index entry is written to the index buffer associated with the write buffer (box 340). In preparation to batch write out the web objects, the cell number of each cell corresponding to web objects is recomputed by adding the value of the next available cell number within the data buffer in persistent cache 210. In one embodiment, after the cell numbers are recomputed, the next available cell number of the disk is then incremented by the number of data blocks occupied by all of the cells in the write buffer. Finally, the write buffer may then be written to the disk in one batch write (box 350). Because of batch writes, objects in a write buffer that are waiting to be written to a disk may be lost in a system crash. This is acceptable because a lost cache object will merely result in a cache miss the next time the web object is requested. In response to the subsequent cache miss, the object will be cached again.

FIG. 4 is a flow diagram illustrating a method, consistent with the present invention, of storing retrieved web objects in a hash table. To begin the process, an object is retrieved from an origin server (box 410). Next, corresponding URL-hash and host-hash values are calculated (box 420). Next, a write buffer is selected based on the host-hash value (box 430). Next, a new hash table entry is created (box 440). Next, the new hash table entry is stored in the hash table based on the URL-hash (box 450).

FIG. 5 is a block diagram of a system, consistent with the present invention, for computing hash values. In one

8

embodiment consistent with the present invention, origin server host identifier 510 is the source of information for computing host-hash 550. In one embodiment, host identifiers may be domain name system ("DNS") host names as conventionally used in the Internet. Host identifier 510 may be, for example, www.sun.com. In order to calculate host-hash 550, host-hash function 530 is applied to host identifier 510. Similarly, in one embodiment, web object URL 520 is used to calculate URL-hash 560. This calculation is performed by applying web-object hash function 540 to URL 520.

FIG. 6 is a block diagram of a hash table consistent with the present invention. In one embodiment, a hash table entry comprises a URL 600 corresponding to a web object, a size 610 of the web object, disk ID 620, and disk location 630, indicating where the web object is stored. If the web object is also stored in memory, the cache entry will also contain memory address 640 corresponding to the address of the web object in the fast main memory cache.

FIG. 7 is a block diagram of a hash table, consistent with the present invention, with URL buckets. In one embodiment, hash table 700 is comprised of multiple buckets such as, for example, URL buckets 710, 720 and 730. Each URL bucket contains entries as described in connection with FIG. 6.

FIG. 8 is a flow diagram of a method, consistent with the present invention, for responding to a request from a web client. First, the method comprises receiving a request for a web object (box 810). The hash table is then searched to locate a hash table entry that has the same URL bucket (box 820). If the hash table entry indicates that the web object is in the memory cache (box 830), the object is returned to the requester (box 840) and the process is complete. Otherwise, it is determined whether the object is stored in persistent memory (box 850). Since many disks can retrieve additional blocks with little additional overhead, an integer number M additional data blocks may be read after the first cell read at the identified disk location (box 860). In one embodiment, not all web objects contained in the extra blocks are placed in the object cache. In this embodiment, only web objects with host identifiers that are the same as the requested web object are placed into the object cache. Web objects that do not have matching identification are ignored for lack of relevance. The hash table entry of each web object that is retrieved from persistent cache in a batch read may then be updated to indicate that the object is now in the fast-main-memory object cache. This may be accomplished by using the URL-hash to select the bucket with the corresponding hash table entry.

The foregoing description of embodiments of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be learned from practicing the invention. For example, the described implementation includes software, but the present invention may be implemented as a combination of hardware and software or in hardware alone. The scope of the invention is defined by the claims and their equivalents.

We claim:

1. In a system having main memory and persistent memory, the main memory containing at least one data buffer, a method for caching related objects, the method comprising:

receiving a plurality of objects from at least one origin server;

9

computing a hash value corresponding to source identifying information for at least one object in the plurality of objects, wherein computing comprises:  
 computing a first hash value based on an object identifier associated with the object server; and  
 computing a second hash value based on a host identifier associated with the origin server; and  
 storing, in at least one data structure of a plurality of data structures, the object based on the computed hash value, wherein the object is stored with related objects having related hash values.

2. A method according to claim 1, wherein the plurality of data structures further comprises:  
 an index buffer capable of storing index entries;  
 an object buffer capable of storing and retrieving objects by logical block number; and  
 a metadata buffer capable of storing information about the index buffer and the object buffer.

3. A method according to claim 2, wherein the index buffer is a cyclic index buffer.

4. A method according to claim 1, wherein the object identifier is a uniform resource locator.

5. A method according to claim 4, wherein the host identifier is a domain-name-service host name associated with the origin server.

6. A method according to claim 1, wherein storing the object using a plurality of data structures further comprises:  
 selecting a write buffer based on the second hash value, the write buffer selected from a plurality of write buffers; and  
 storing information about the object in a hash table using the first hash value.

7. A method according to claim 6, wherein the write buffer is contained in fast main memory.

8. A method according to claim 6, wherein the persistent memory comprises a disk storage device.

9. A method according to claim 5, wherein storing the object further comprises:  
 selecting a write buffer based on the second hash value, the write buffer selected from a plurality of write buffers; and  
 storing information about the object in a hash table using the first hash value.

10. A method according to claim 6, further comprising:  
 selecting a persistent storage device on which to write objects to be written that are stored in fast main memory;  
 retrieving hash table entries corresponding to the objects to be written; and  
 modifying the hash table entries to indicate that the objects to be written are stored in persistent storage.

11. A method according to claim 10, wherein the persistent storage device is a disk storage device.

12. A method according to claim further comprising:  
 selecting a persistent storage device on which to write objects to be written that are stored in fast main memory;  
 retrieving hash table entries corresponding to the objects to be written; and  
 modifying the hash table entries to indicate that the objects to be written are stored in persistent storage.

13. A method according to claim 12, wherein the persistent storage device is a non-volatile memory storage device.

14. A method according to claim 10, further comprising:  
 storing information about the objects to be written in the index buffer; and batch writing the objects to be written to persistent storage.

10

15. In a system having main memory and persistent memory, the main memory containing at least one data buffer, a method for retrieving cached related objects comprising:

receiving a request for an object, the request comprising an object identifier;  
 looking up the requested object in a hash table using the object identifier;  
 determining whether the object is stored in persistent memory;  
 retrieving a group of related objects from an object buffer capable of storing and retrieving objects by logical block number, when it is determined that the object is stored in persistent memory; and  
 discarding those retrieved objects that do not have an origin server host identifier in common with the requested object.

16. A method according to claim 15, wherein receiving a request for an object further comprises receiving a request from a web client for a web object.

17. A method according to claim 15, wherein the object identifier is a uniform resource locator.

18. A method according to claim 15, wherein retrieving a group of related objects from an object buffer further comprises:

reading an integer number of logical blocks in a single read operation.

19. A computer system having main memory and persistent memory, the main memory containing at least one data buffer, and an execution unit capable of executing program code, the computer system comprising:

a network interface capable of receiving information comprising a plurality of objects from at least one host computer;

a controller configured to calculate a hash value corresponding to source identifying information for at least one object in the plurality of objects by at least computing a first hash value based on an object identifier associated with the object, and computing a second hash value based on a host identifier associated with the origin server; and

a memory used to store, in at least one data structure of a plurality of data structures, the object based on the hash value, wherein the object is stored with related objects having related hash values.

20. A computer system comprising:

means for receiving data comprising a plurality of objects from at least one origin server;

means for computing a hash value corresponding to source identifying information for at least one object in a plurality of objects, wherein computing comprises:  
 computing a first hash value based on an object identifier associated with the object, and  
 computing a second hash value based on a host identifier associated with the origin server; and

storing, in at least one data structure of a plurality of data structures, the object based on the computed hash value, wherein the object is stored with related objects having corresponding hash values.

21. A computer-readable medium containing program code capable of causing a computer system to execute a method for caching related objects, the computer-readable medium comprising:

program code operative to receive a plurality of objects from at least one origin server;

## 11

program code operative to compute a hash value corresponding to source identifying information for at least one object in the plurality of objects by at least computing a first hash value based on an object identifier associated with the object, and computing a second hash value based on a host identifier associated with the origin server; and

program code operative to store, in at least one data structure of a plurality of data structures, the object based on the computed hash value, wherein the object is stored with related objects having corresponding hash values on a disk storage device.

22. A computer-readable medium according to claim 21, wherein the program code operative to store the object further comprises:

program code operative to select a memory region based on the second hash value, the memory region selected from a plurality of memory regions; and

program code operative to store information about the object in a hash table using the first hash value.

23. A computer-readable medium according to claim 22, further comprising:

program code operative to select a disk on which to write information comprising a plurality of objects from fast main memory;

program code operative to retrieve hash table entries corresponding to the objects to be written; and

program code operative to modify the hash table entries to indicate that the objects to be written are stored on disk.

24. A computer-readable medium according to claim 23, further comprising:

## 12

program code operative to store an identification of the objects to be written in an index buffer; and

program code operative to cause the objects to be written to disk in at least one batch write operation.

25. A computer-readable medium having main memory and persistent memory, the main memory containing at least one data buffer, and an execution unit capable of executing program code to perform a method for retrieving cached related objects, the computer-readable medium comprising:

program code operative to receive a request for an object, the request comprising an object descriptor;

program code operative to identify the requested object based on the object descriptor, using a hash table;

program code operative to determine whether the object is stored on disk;

program code operative to accept a group of related objects from an object buffer, when it is determined that the object is stored on disk; and

program code operative to discard those objects that do not have an origin server host identifier in common with the requested object.

26. A computer-readable medium according to claim 25, wherein the program code operative to accept a group of related objects from an object buffer further comprises:

program code operative to batch read information from disk.

\* \* \* \* \*



US006598119B2

(12) **United States Patent**  
Becker et al.

(10) Patent No.: **US 6,598,119 B2**  
(45) Date of Patent: **Jul. 22, 2003**

(54) **DATABASE MANAGEMENT SYSTEM WITH  
A MULTIPLE-LEVEL CACHE  
ARRANGEMENT**

(75) Inventors: **Richard Alan Becker**, Mendham  
Township, Morris County, NJ (US);  
**Allan Reeve Wilks**, Scotch Plains, NJ  
(US)

(73) Assignee: **AT&T Corp.**, New York, NY (US)

(\*) Notice: Subject to any disclaimer, the term of this  
patent is extended or adjusted under 35  
U.S.C. 154(b) by 281 days.

(21) Appl. No.: **09/780,633**

(22) Filed: **Feb. 9, 2001**

(65) **Prior Publication Data**

US 2002/0112123 A1 Aug. 15, 2002

(51) Int. Cl.<sup>7</sup> ..... **G06F 12/00**

(52) U.S. Cl. .... **711/119; 711/141; 711/165;  
705/1; 705/2; 705/5; 707/201; 707/104.1**

(58) Field of Search ..... **705/1, 2, 3; 707/201,  
707/202, 205, 104.1, 102; 711/119, 122,  
124, 126, 133, 134, 135, 141, 142, 143,  
165**

(56) **References Cited**

#### U.S. PATENT DOCUMENTS

5,553,265 A \* 9/1996 Abato et al. .... 711/143  
5,627,993 A \* 5/1997 Abato et al. .... 711/143  
5,628,014 A \* 5/1997 Cecchini et al. .... 707/205  
5,778,422 A \* 7/1998 Genduso et al. .... 711/117

5,809,530 A \* 9/1998 Samra et al. .... 711/140  
5,909,697 A \* 6/1999 Hayes et al. .... 711/144  
5,963,963 A \* 10/1999 Schmuck et al. .... 707/205  
6,073,212 A \* 6/2000 Hayes et al. .... 711/122  
6,321,304 B1 \* 11/2001 James ..... 711/141  
6,438,659 B1 \* 8/2002 Bauman et al. .... 711/141  
2002/0103976 A1 \* 8/2002 Steely et al. .... 711/135

#### OTHER PUBLICATIONS

Richard A. Becker, "Fundamentals of Data Structures,"  
Computer Science Press, 1976, pp. 496-517.

\* cited by examiner

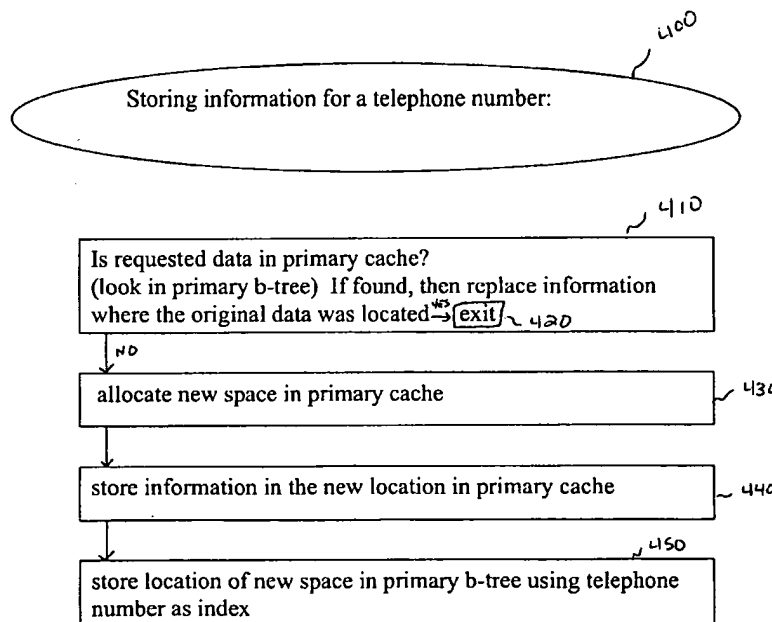
Primary Examiner—Donald Sparks

Assistant Examiner—C. P. Chace

(57) **ABSTRACT**

A data management system for storing data in a multiple-level cache arrangement of a database comprises a multi-tier cache memory for initially storing all data in summary form in a secondary cache which may be the database; a processor for receiving requests for data and for moving requested data from the secondary cache to a primary cache, wherein, when subsequent requests for data are received, the primary cache is searched before the secondary cache; and for periodically synchronizing and merging all data in the primary cache back into said secondary cache to refresh said primary cache and remove stale information. The system is particularly useful for managing a telecommunications system call detail summary database in which telephone call details are collected as AMA records after the calls terminate and the AMA records are forwarded to a call detail database for storage in summary form and analysis by an external system, for example, for fraud analysis or billing purposes.

**18 Claims, 7 Drawing Sheets**



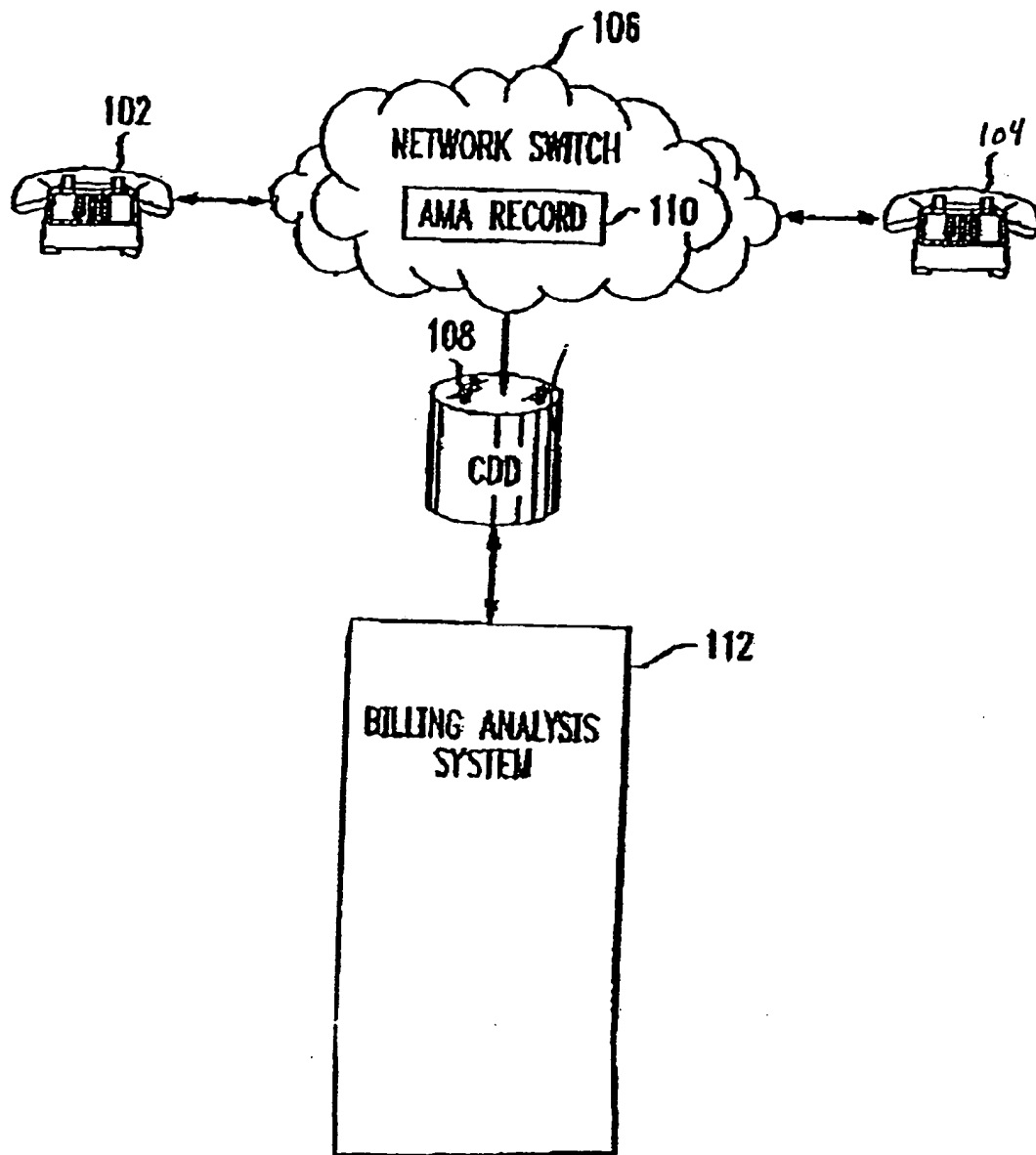


Fig. 1

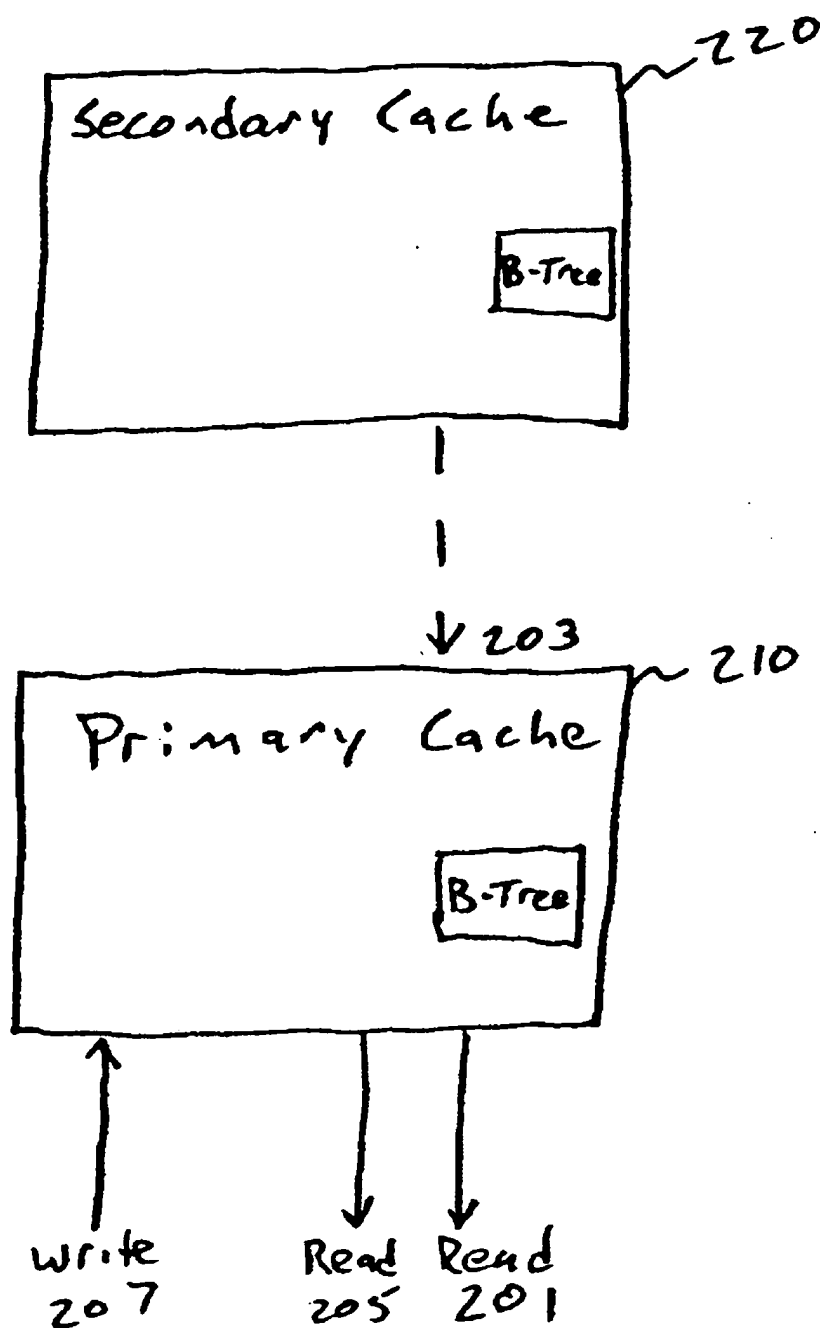


Fig. 2

FIG. 2A

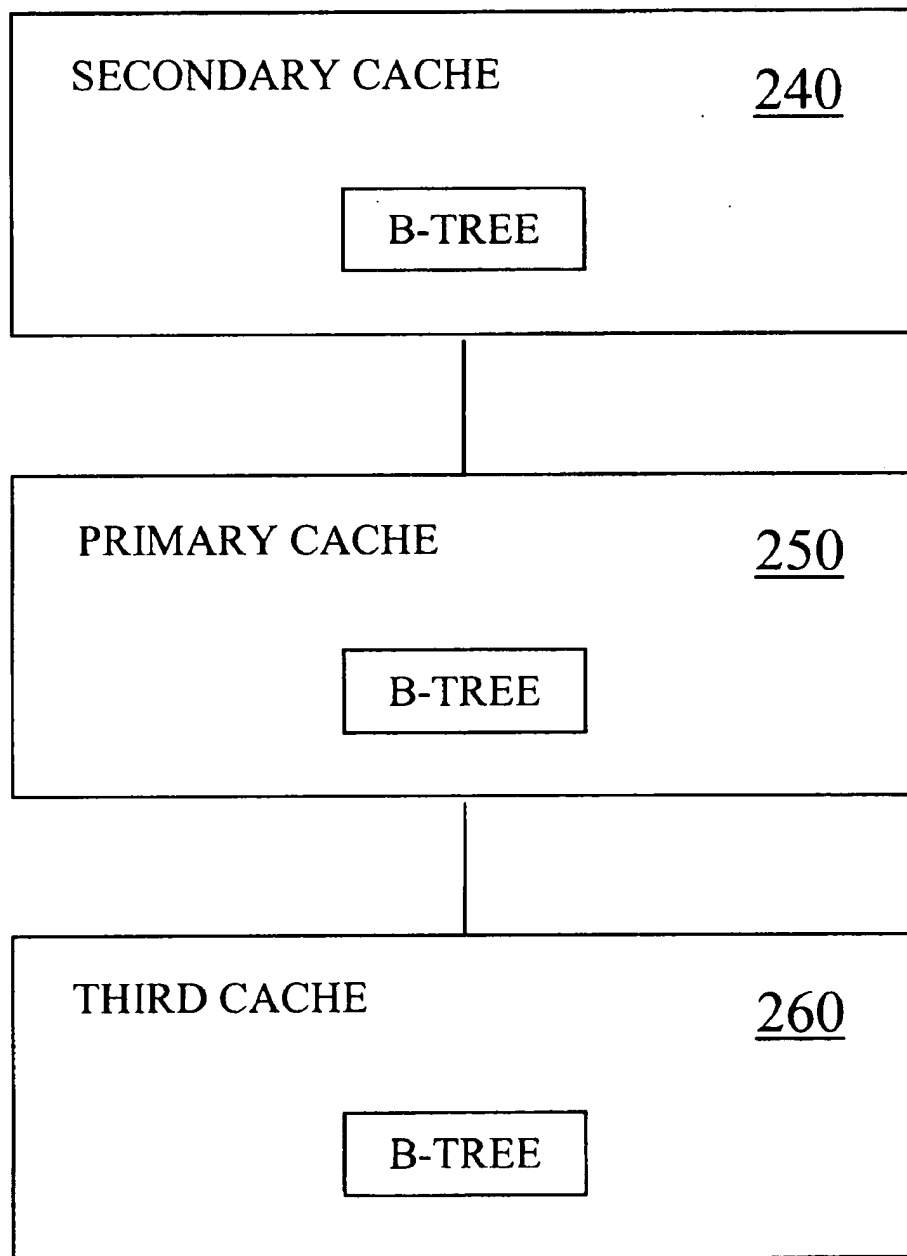


FIG. 2B

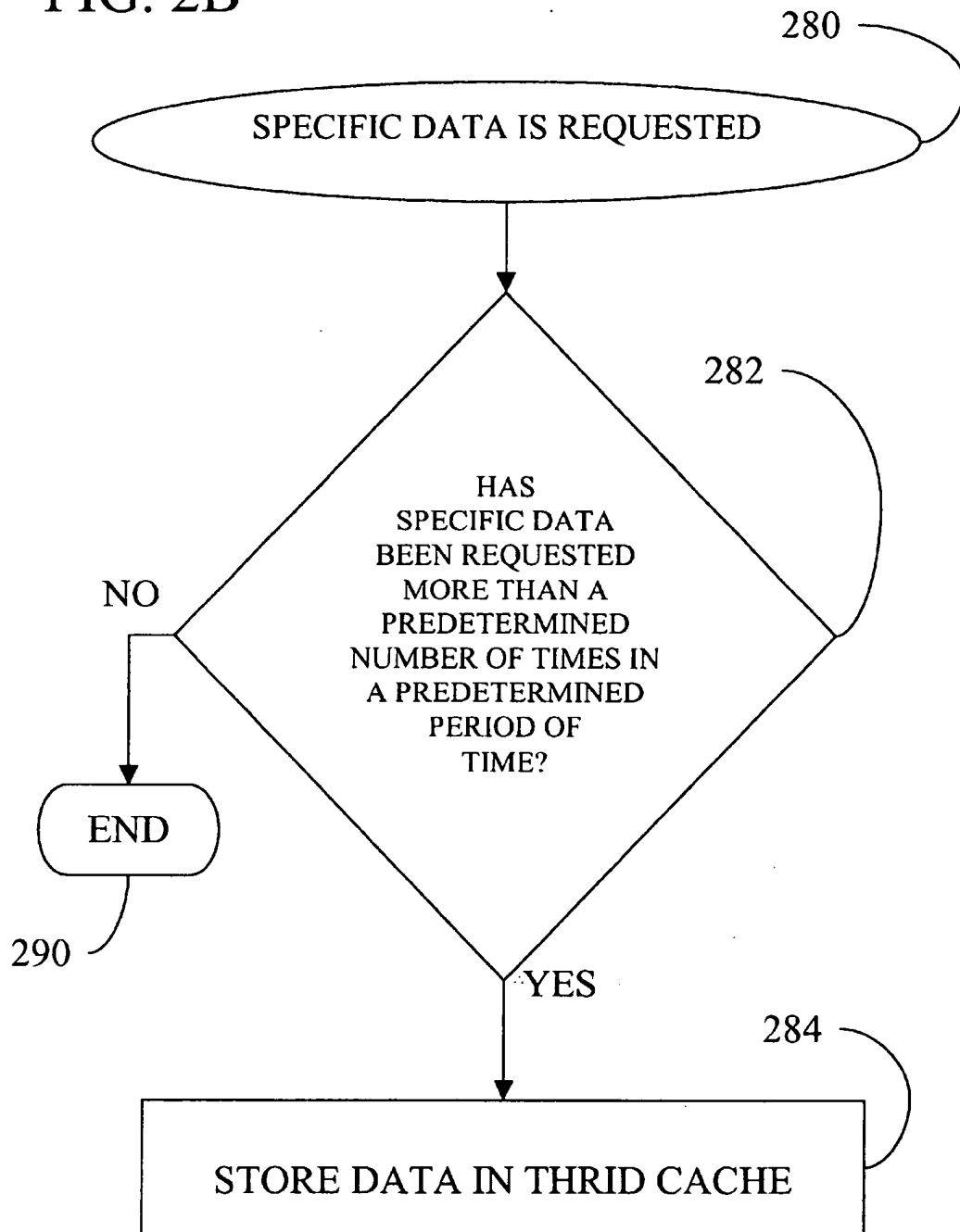


Fig. 3

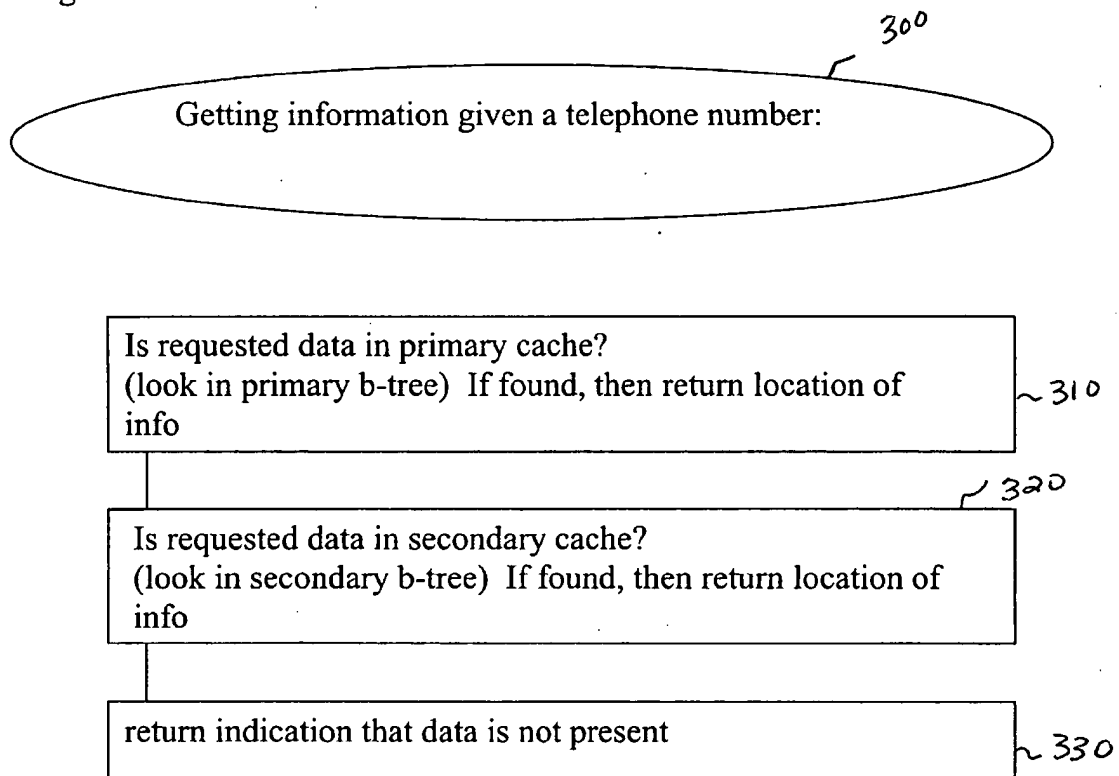


Fig. 4

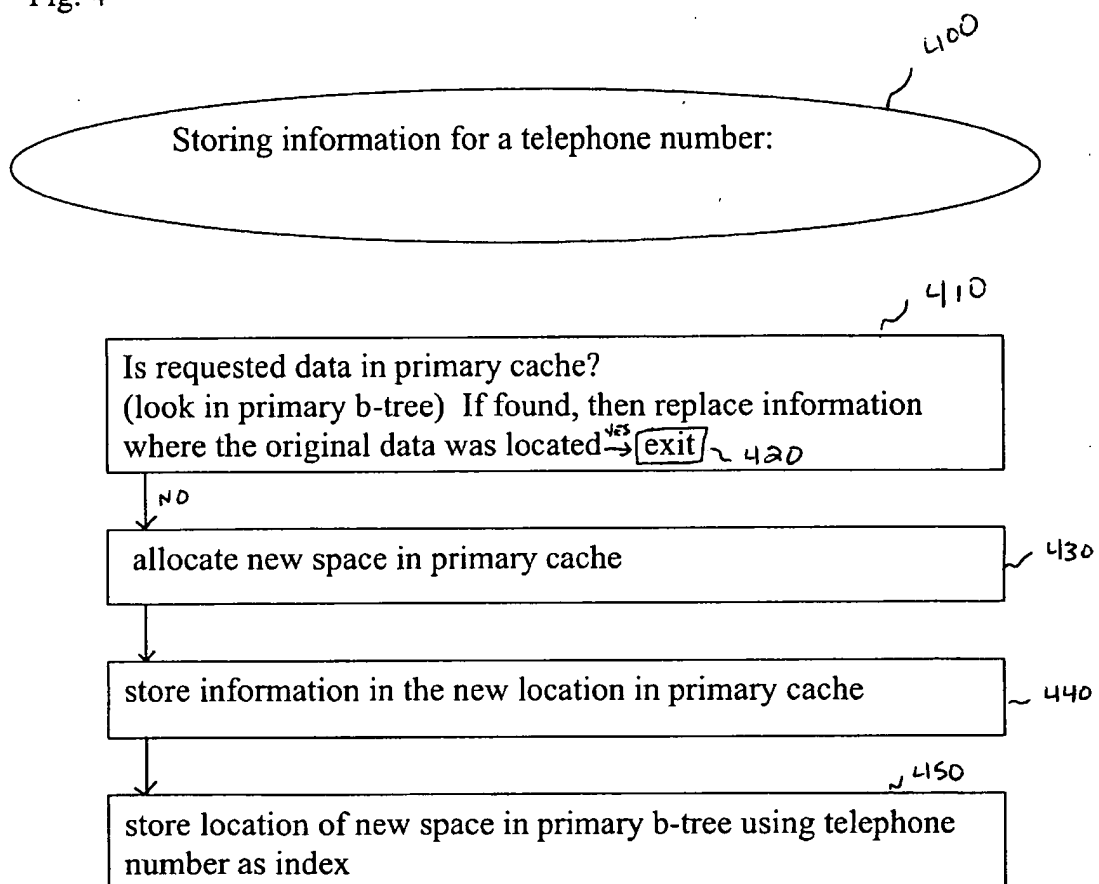
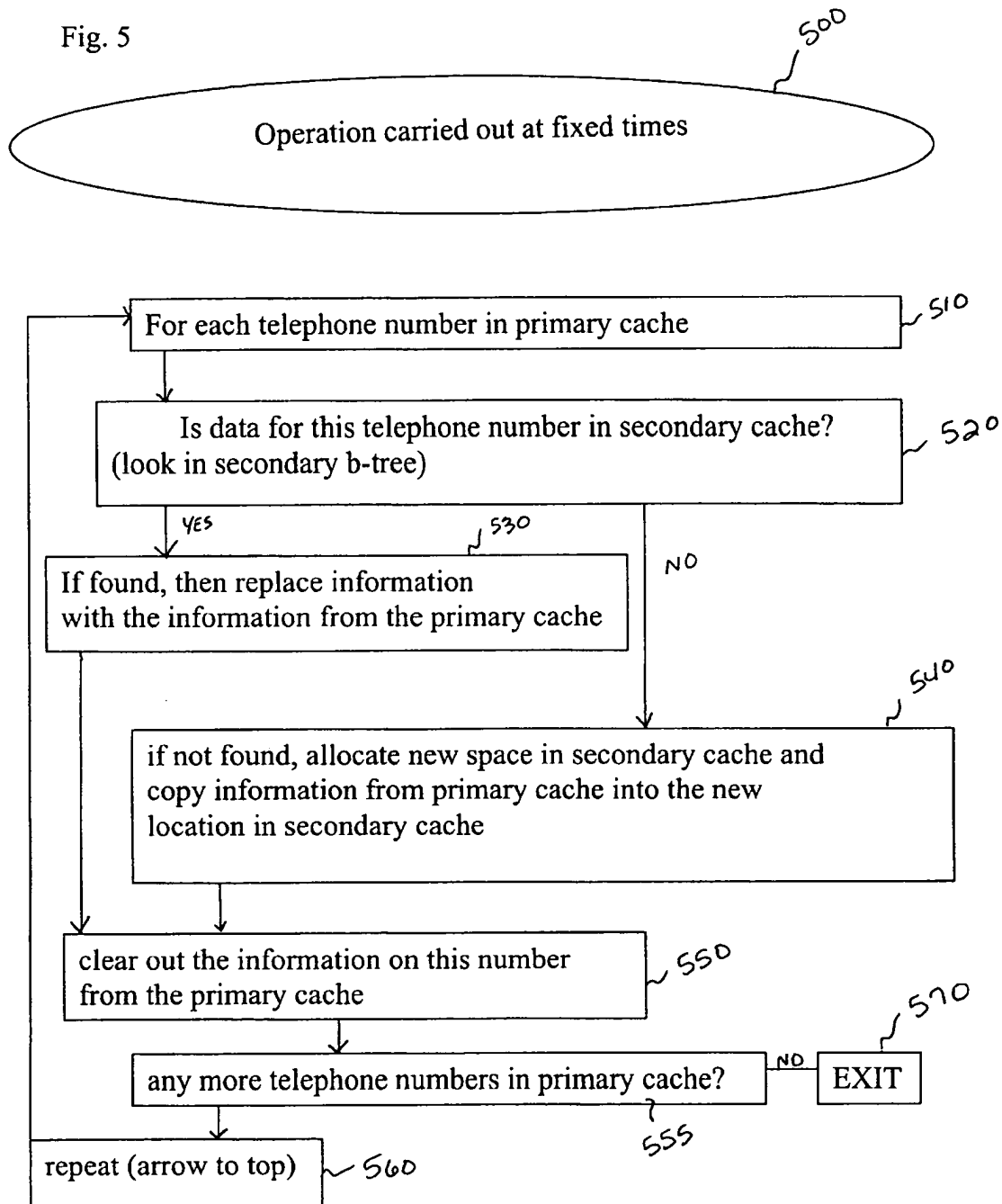


Fig. 5



1

# **DATABASE MANAGEMENT SYSTEM WITH A MULTIPLE-LEVEL CACHE ARRANGEMENT**

## **FIELD OF THE INVENTION**

The invention relates to a method and apparatus for storing information, and particularly to a data storage system which uses a multiple-level cache arrangement wherein data may be initially in a secondary cache and is moved from a secondary cache to a primary cache when the data is first requested by a system.

## **BACKGROUND OF THE INVENTION**

Because immediate access to information has become a necessity in virtually all fields of endeavor, including business, finance and science, telecommunication system usage is increasing at a substantial rate. With the increase in overall usage, the amount of information generated by the telecommunication systems has also exploded. Typically, each call made on a telecommunication system generates a variety of information, for example, billing and other information. For example, one telephone call may involve an Internet service provider, a wireless carrier, a local exchange carrier and a long-distance carrier among other entities. Each of these entities may have its own way of pricing their portion of a telephone call involving equipment charges, service charges, access charges and other charges appearing on one or several bills to a subscriber. All of this information can be analyzed for a variety of purposes such as billing, fraud control, detecting trends and characteristics of each telephone number (directory number or other billing number or indicia of a subscriber) in a telecommunication system. For each of these purposes, a processor in the analysis system accesses the call information, e.g., call detail, specifying various call parameters. For example, the analysis system may want to analyze the 100 most recent calls for a particular telephone number. The call detail information, which is stored in the telecommunication system, is then gathered from the database and sent to the analysis system requesting the information. The gathering process takes time and resources. Furthermore, as the amount of information in storage increases, the amount of time needed to search for requested information also increases. Thus, there is a need for a data management system which efficiently stores data in a manner that allows for faster searching for requested data.

## **SUMMARY OF THE INVENTION**

According to one embodiment of the invention, a method and data management system stores data in a multiple-level cache arrangement of a database. Data may be initially stored in a secondary cache (which may be the database). When data is requested, the data is moved from the secondary cache to a primary cache. When a new search for data is initiated, the search begins with the primary cache and then proceeds to the secondary cache, if necessary.

According to one embodiment of the invention in a telecommunications system application, a method and apparatus for storing data in a multiple-level cache arrangement of a database is disclosed. Initially, all data for telephone numbers is stored in a secondary cache (which may be the database). When a request for data is received, the requested data is moved from the secondary cache to a primary cache, wherein when subsequent requests for data are received, the primary cache is searched before the secondary cache.

2

Finally, all data in the primary cache is periodically merged back into the secondary caches so that both the primary and secondary cache reflect similar information and old data may be purged.

5

## **BRIEF DESCRIPTION OF THE DRAWINGS**

The foregoing summary of the invention, as well as the following detailed description of the preferred embodiments, is better understood when read in conjunction with the accompanying drawings, which are included by way of example and not by way of limitation with regard to the claimed invention.

FIG. 1 illustrates a block diagram of a telephone system which can implement embodiments of the invention;

15

FIG. 2 illustrates a multiple-level cache arrangement according to one embodiment of the invention;

FIG. 2A illustrated a multiple-level cache arrangement according to one embodiment of the invention;

20

FIG. 2B is a flowchart illustrating one example of the operation of storing data to a third cache;

FIG. 3 is a flow-chart illustrating the operation of one embodiment of the invention for obtaining information for a given telephone number,

25

FIG. 4 is a flowchart illustrating the operation of one embodiment of the invention for storing information for a telephone number; and

30

FIG. 5 is a flowchart illustrating the operation of one embodiment of the invention for periodically updating or merging information of primary and secondary caches for each telephone number.

## **DETAILED DESCRIPTION**

The invention relates to a data management system which stores data in an efficient manner allowing faster searching of data. The data management system is particularly effective in systems in which large amounts of data need to be stored and a small amount of stored data is repeatedly requested. One such system is a telephone system like the exemplary system illustrated in FIG. 1. It will be understood that the invention is not limited to being implemented in a telephone system but rather the invention can be implemented in many other types of systems.

According to one embodiment of the invention, the data is divided into a plurality of categories—for example, two. Data can be referred to as inactive or quiet if the data has not been requested within a predetermined period of time, and data can be referred to as active if the data has been requested within the predetermined period of time. Data is initially stored in a secondary cache which may be a database. As will be explained below with reference to FIGS. 3–5, data is moved from a secondary cache for quiet data to another cache, a primary cache for active data, when data is requested, and subsequent searches for data are then first performed using the primary cache containing active data and then performed on the secondary cache containing quiet or inactive data if necessary. It is important to note that the invention is suited for a situation in which the different pieces of data in the secondary cache are requested at different rates. For example, in a telephone system, if everyone made one telephone call per day, then the primary and secondary caches would become the same size. However, it has been observed that a small number of the total number of telephones account for a large number of all of the calls made during a day.

Referring to FIG. 1, there is shown a block diagram of an exemplary telephone system which can implement the

3

invention. There is shown a calling telephone 102, a called telephone 104, a telephone network switch 106 and a Call Detail Database (CDD) 108 for storing call detail data in summary form as will be further described herein. An Automatic Message Accounting (AMA) record, represented by a block 110, is also shown. As indicated by FIG. 1, a billable call may be initiated at telephone 102 and routed through switch 106, e.g., a 4ESS switch, now manufactured by Lucent Technologies, Inc., to telephone 104. The switch 106 generates AMA record 110, which includes the information necessary to price the call. Typical AMA information includes a start or termination time of day and date, a directory or calling number and a called number, among other call details. The AMA record is passed to the CDD 108 for summary and storage by telephone number as an index. It should be noted here that there are an abundance of protocols and transmission media that may be used for passing the data from the switch to the CDD. For example, suitable protocols include the well known File Transfer Protocol (FTP) and Transmission Control Protocol/Internet Protocol; and suitable transmission media include twisted shielded pair wires, fiber optic lines, coaxial cable, and wireless links. Moreover, these protocols and media are suitable for use in all data transfers and queries hereinafter described.

In any event, once the AMA record has been passed to the CDD, it may be summarized and stored, indexed by telephone number as will be described herein and may be made available for use in pricing the call or for other purposes such as fraud analysis. To this end, a summary record in response to a query is passed to a billing or other analysis system 112, which may be a general purpose computer capable of running the software necessary to implement the invention. The present invention, however, relates to a database management system implemented more typically at CDD 108 to manage and provide efficient data storage.

An analysis system 112 can be a billing analysis system, a fraud control system and/or some other system which is used to analyze call detail summary information. The analysis system 112, in the billing analysis example, applies any customer-specific billing parameters to the AMA record summary to produce a processed AMA record summary. It then may pass the AMA record summary and the processed AMA record summary back to the CDD for storage.

According to one embodiment of the invention, a call detail record is received at CDD 108, where a host processor for CDD 108 accepts the record and identifies the telephone number associated with the record. The CDD system according to the present invention asks to see the summary of that telephone number's usage already in CDD 108. That record summary, referred to herein as a call detail record summary, is retrieved from storage using the multiple-level cache structure—for example, two tiers which may include the database itself, as will be discussed with reference to FIG. 2. The retrieved summary is read, updated and rewritten into the primary cache, even if it is retrieved from a secondary cache. This action ensures that the updated record will be quickly accessible on the next access. Thus, the present invention provides as fast an access as possible in a telecommunications network setting that is handling millions of telephone calls per day.

In keeping with the present invention, all of the billing information can be stored in a secondary (quiet) cache, which may be the database itself, at the beginning of each day. Then as specific billing data for particular subscribers is requested, for example, by the billing system or when new data comes in, the requested data record is moved from the

4

secondary cache to a primary (active) cache. The active data then stays in the primary cache for a predetermined period of time—for example, an hour, a day, a week, etc.—at which point all of the data in the primary cache are merged back into the secondary cache, i.e., information from the primary cache is copied back into the secondary cache, removed from the primary cache and old information in the secondary cache may be dropped or revised in some known manner. These data processing details will be further explained with reference to FIGS. 3–5.

In one embodiment of the invention, two levels of caches are shown: primary and secondary, as shown in FIG. 2, where the secondary cache may be the database itself. It will be understood that more than two levels of caches can be used in this invention and the invention is not limited to a specific number of levels. For example, three levels of caches can be used: secondary, primary, and a third active cache. In this example, if specific data is requested more than a predetermined number of times during a predetermined period of time, the data can be moved from the primary cache to the third cache.

Referring to FIG. 2A, a three-tier illustrative embodiment is shown. The three-tier arrangement comprises a secondary cache 240, a primary cache 250, and a third cache 260. In particular, FIG. 2B shows a process for storing data in a third cache as shown in FIG. 2A. At specific data has been requested. At step 282, a determination is made as to whether the specific data has been requested more than a predetermined number of times during a predetermined period of time. If not, the process is exited at step 290. On the other hand, if the specific data has been requested more than a predetermined number of times during a predetermined period of time, at step 284, the data can be moved from the primary cache to the third cache.

In a two-tier embodiment, the secondary cache is synchronized each day at “synchronization” time and the data are marked read-only. By “synchronization” time herein is intended the time when some data is determined to be outdated for merging or otherwise determined to require replacement, movement from one cache to another or clearing or removal from cache memory altogether due, for example, to data staleness. For example, a telephone number may cease to appear in AMA records when a subscriber disconnects their telephone service and their associated data become stale.

The primary cache, a location of active data, is based on any activity since the synchronization process. Initially, its size is determined to be large enough to hold summary data collected from one synchronization time to the next. Thus it may be considered to be variable in size and is periodically cleared. This allows the primary cache to be much smaller than the entire universe of subscribers that are monitored.

Known cache memory is typically size-limited and information is typically migrated out of such cache memory when the memory approaches fullness or is full, which would occur asynchronously. The primary cache memory of the present invention is cleared out periodically or synchronously at synchronization time but not necessarily at regular intervals. Moreover, one time of day for synchronizing the primary and secondary caches may be intentionally chosen to be a non-busy hour, for example, in the middle of the night when traffic volume is at a minimum. The present invention, however, contemplates a combination of asynchronous (when one cache approaches a fullness state) and synchronous reaping (for example, every non-busy hour).

In one embodiment of the present invention, B-trees are used to access disc blocks already cached by the operating

5

system. In alternative embodiments, other means for indexing data for retrieval may be utilized, and use of a B-tree is not critical to the present invention but is exemplary and is further discussed below. There exist several descriptions of B-trees; one such description may be found at Chapter 10 of *Fundamentals of Data Structures*, Computer Science Press, Potomac, Md., U.S.A., 1976.

In a B-tree embodiment, the telephone number as defined above is used as a key or index to access information regarding the telephone number. When information regarding a new call comes in, the call detail summary information already recorded regarding the telephone number is looked up. As illustrated in FIG. 2, a read operation 201 is first performed on the primary cache 210. If the desired information is not in the primary cache 210, a read operation 203 is then performed on the secondary cache 220. A B-tree (not specifically referenced with a reference numeral but shown) associated with each cache takes the telephone number as an input and outputs the location of the call detail summary information.

In a known database management system, a B-tree index is one means for rapidly and efficiently accessing each record in a database. A B-tree index has one apex root node and a plurality of nodes divided into branches at a number of hierarchical levels, diverging from the root node. Of the branched nodes, those nodes at the lowest level are often called leaf nodes. Those nodes other than the leaf nodes are often called upper nodes. The upper nodes include the root node. Each node has a plurality of index entries constituted by the following data. A leaf node entry typically has a key value and a pointer to the record in the database associated with the key value. An upper entry node has a pointer to a child node at the next lower level and one key value representative of the range of key values covered in the leaf nodes branched from the child node. The key value in an upper index entry functions as a guide (decision element) by which an access program searches in the B-tree index from the root node to the leaf node which has the index entry including the pointer to the target record.

Returning to a discussion of FIG. 2, the call detail summary information is read at 205. The information can then be updated and rewritten 207 back into the primary cache 210 regardless of where it is read from. When the information is read, typically the entire disk block containing that information is read. Unfortunately, disk reads are slow, so if the system is trying to keep up with a fast real-time feed of calls, the system is limited by the time it takes to read and rewrite the disk blocks. Thus, it is desirable to have a smaller number of disk blocks stored in the primary cache 210 and to have data initially written to a secondary cache until accessed.

FIGS. 3-5 are flow charts depicting the operation of one of the embodiments of the invention which uses a two level cache arrangement: primary (active) cache and secondary (inactive) cache which may be the database. Initially, all of the data is stored in the secondary cache. As data is requested, the requested data is moved from the secondary cache to the primary cache. Likewise, as new data is created, the new data is stored in the primary cache. As a new request for data is received, the primary cache is first searched to see if it contains the requested data. If the requested data is not in the primary cache, the secondary cache is searched. Finally, at a predetermined interval, the data in the primary cache is merged back into the secondary cache.

In particular, FIG. 3 shows a process for looking up information for a given telephone number in the primary and secondary caches shown in FIG. 2. At step 310, the system asks whether the requested data is in the primary cache 210 by looking in the primary B-tree (unlabeled). If found, then the location of the information is returned to the system in

6

response to the telephone number request. At step 320, the system then asks whether the requested data for the telephone number is in the secondary cache 220 by looking in the secondary B-tree (unlabeled). If found, then the location of the information is returned to the system in response to the telephone number request. Thus, the system first looks to the primary cache and then to the secondary cache for information indexed by telephone number.

FIG. 4 refers to a process for storing information for a given telephone number after the initial process of loading a secondary cache with initial information. At step 410, the system asks whether the requested data is in the primary cache 210 (again, by looking in the primary B-tree). If found, then the information at the location where the original data is stored is replaced with the new or updated information. Then, the process is exited at step 420. On the other hand, if the requested data is not in the primary cache, then at step 430 new memory space is allocated in primary cache 210 for the data. At step 440, the information is stored in the new location in primary cache 210. Finally, so the data can be retrieved, the location of the new space is stored at step 450 in the primary B-tree using the telephone number as the index or key.

FIG. 5 shows a process of periodically merging or otherwise assuring data integrity and continuity. In a preferred embodiment, the process occurs synchronously or periodically and at a time of day when there is little traffic such as in the middle of the night. The process may also be practiced aperiodically, for example, when one cache memory achieves a certain degree of fullness. The process is carried out for each telephone number index into the primary cache at step 510. The system asks at step 520, is the data for this telephone number in the secondary cache 220? To do so, the secondary B-tree is looked to for this information. If the answer is "yes", then at step 530 the system replaces information with the information from the primary cache. If the answer is no, at step 540, and the information is not found in the secondary cache (where it should be), new space is allocated in the secondary cache 220 for the information and information is copied from the primary cache into the new location in the secondary cache. Both steps 530 and 540 follow a path to step 550 where information is cleared out of memory regarding the telephone number index from the primary cache. Then, at step 555, the question is asked whether there are any more telephone numbers not processed in primary cache. If there are more telephone numbers to process in the primary cache, then, per step 560, the process is repeated for each remaining telephone number in the primary cache and then the process is repeated again, for example, each fixed time interval selected by the system. If this was the last telephone number during this fixed time interval, then the process is exited at box 570 until the next fixed time interval.

Despite the fact that primary cache memory is sized to be able to handle summary data from one reaping process to the next, there is some possibility that the primary cache memory may approach a full state. In such a situation, it may be appropriate to combine the process described in FIG. 5 with a primary cache memory status monitor such that when the primary cache memory reaches a predetermined degree of fullness, memory may be cleared in the primary cache by performing the process of FIG. 5 during the fixed time interval. For example, when the memory status monitor signals the primary cache is reaching a fullness state, the process of FIG. 5 is invoked. Alternative approaches to such a combination of synchronous and asynchronous reaping may also come to mind in one of ordinary skill in the art.

Although preferred embodiments of the method and apparatus of the invention have been illustrated in the accompanying Drawings and described in the foregoing Detailed

7

Description, it is understood that the invention is not limited to the embodiments disclosed but is capable of numerous rearrangements, modifications, and substitutions without departing from the spirit or scope of the invention as set forth and defined by the following claims.

We claim:

1. A method for storing data in a multiple-level cache arrangement, comprising the steps of:  
initially storing all data in a secondary cache;  
receiving a call at a network switch;  
forwarding Automatic Message Accounting records from said network switch to a call detail database for storage as a call detail record summary indexed by telephone number;  
receiving requests for data;  
moving requested data from said secondary cache to a primary cache, wherein when subsequent requests for data are received, the primary cache is searched before the secondary cache;  
updating said summary of Automatic Message Accounting records; and  
periodically synchronizing and merging data in said primary cache back into said secondary cache to refresh said primary cache and remove stale information, wherein said data comprises call detail data indexed by telephone number,  
wherein said call detail data comprises a summary of Automatic Message Accounting records for a telephone number for storage in said call detail database.
2. The method for storing data according to claim 1, wherein all data in said secondary cache is read-only data.
3. The method for storing data according to claim 1, wherein data in said primary cache can be amended.
4. The method according to claim 1, further comprising the step of:  
storing requested data in a third cache if said data has been requested more than a predetermined number of times in a predetermined period of time.
5. A data management system for storing data in a multiple-level cache arrangement, comprising:  
means for initially storing all data in a secondary cache;  
means for receiving a call at a network switch;  
means for forwarding Automatic Message Accounting records from said network switch to a call detail database;  
means for storing forwarded Automatic Message Accounting records as a call detail record summary indexed by telephone number;  
means for receiving requests for data;  
means for moving requested data from secondary cache to a primary cache, wherein when subsequent requests for data are received the primary cache is searched before the secondary cache;  
means for updating said summary of Automatic Message Accounting records; and  
means for periodically synchronizing and merging all data in said primary cache back into said secondary cache to refresh said primary cache and remove stale information,  
wherein said data comprises call detail data indexed by telephone number,  
wherein said call detail data comprises a summary of a plurality of Automatic Message Accounting records indexed by said telephone number for storage in said call detail database for operation upon by one of fraud analysis and call billing programs.

8

6. The data management system for storing data according to claim 5, wherein all data in said secondary cache is read-only data.

7. The data management system for storing data according to claim 5, wherein data in said primary cache can be amended.

8. The data management system for storing data according to claim 5, further comprising:

means for storing requested data in a third cache if said data has been requested more than a predetermined number of times in a predetermined period of time.

9. The data management system according to claim 5, wherein a switch collects an Automatic Message Accounting record for a given call and forwards said record comprising at least a time of day, a date and a telephone number to said call detail database for storage in summary form as a call detail record summary indexed by telephone number.

10. In a method for storing call detail data in summary form in a multiple-level cache arrangement comprising a primary cache and a secondary cache, a synchronization process comprising the steps of:

initially storing call detail data in summary form in the secondary cache;

for each telephone number in the primary cache, looking in the secondary cache for data;

if found, replacing data in the secondary cache with data from the primary cache and, if not found, allocating new space in the secondary cache and copying the data from the primary cache into the new space in the secondary cache;

clearing out data for the telephone number from the primary cache; and

repeating the looking step, one of the replacing and the space allocation and copying steps, and the data clearing step at a predetermined time.

11. A method as recited in claim 10, comprising the initial step of sizing said primary cache to be large enough to hold said summary call detail data from one synchronizing process to the next.

12. A method as recited in claim 10, wherein said synchronizing process further comprises the step of determining if said primary cache is approaching a predetermined degree of fullness.

13. A method as recited in claim 10, wherein said synchronizing process is scheduled at predetermined intervals measured in days and for a time of day when telecommunications traffic is less busy.

14. A method as recited in claim 13, further comprising the step of invoking, responsive to said primary cache approaching a predetermined degree of fullness, a synchronizing process excluding step of storing data in primary cache.

15. The method for storing data according to claim 1, further comprising a step of periodically removing stale information from said secondary cache.

16. The method for storing data according to claim 15, wherein the steps of synchronizing and removing occur simultaneously.

17. The method for storing data according to claim 15, wherein the step of periodically removing stale information from said secondary cache occurs at a time of day when telecommunications traffic is less busy.

18. The data management system for storing data according to claim 5, further comprising means for periodically removing stale information from said secondary cache.

\* \* \* \* \*

UNITED STATES PATENT AND TRADEMARK OFFICE  
**CERTIFICATE OF CORRECTION**

PATENT NO. : 6,598,119 B2  
DATED : July 22, 2003  
INVENTOR(S) : Becker et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

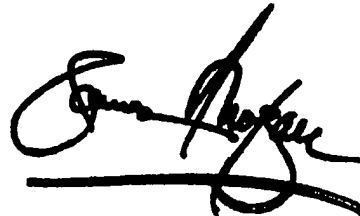
Drawings.

Sheet 1 of 7 through Sheet 7 of 7,

The above-identified Sheet 1 of 7 through Sheet 7 of 7 contain informal drawings for the above-identified patent. These sheets should have contained the formal drawings, Figures 1, 2, 2A, 2B, 3, 4, and 5, which have been approved by the Examiner by the Notice of Allowability mailed May 6, 2003. In particular, this Notice indicates that the drawings filed on February 9, 2001 and April 15, 2003 are accepted by the Examiner. Figures 2A and 2B were formal drawings submitted on April 15, 2003 and have been approved by the Examiner at page 2 of the Notice. The above-identified issued patent contains formal drawings for Figures 2A and 2B (Sheet 3 of 7 and Sheet 4 of 7). However, the originally filed drawings, Figures 1, 2, 3, 4, and 5 that were submitted on February 9, 2001, are informal. Three sheets of formal drawings that contained Figures 1, 2, 3, 4, and 5 were submitted on April 27, 2001.

Signed and Sealed this

Seventh Day of October, 2003

A handwritten signature in black ink, appearing to read "James E. Rogan", with a horizontal line drawn underneath it.

JAMES E. ROGAN  
*Director of the United States Patent and Trademark Office*

UNITED STATES PATENT AND TRADEMARK OFFICE  
**CERTIFICATE OF CORRECTION**

PATENT NO. : 6,598,119 B2  
DATED : July 22, 2003  
INVENTOR(S) : Richard Alan Becker and Allan Reeve Wilks

Page 1 of 1

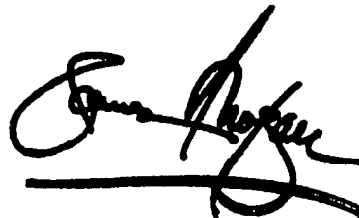
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 8,

Line 5, "inlaid" is replaced with -- in said --

Signed and Sealed this

Fourteenth Day of October, 2003

A handwritten signature in black ink, appearing to read "James E. Rogan", with a horizontal line drawn underneath it.

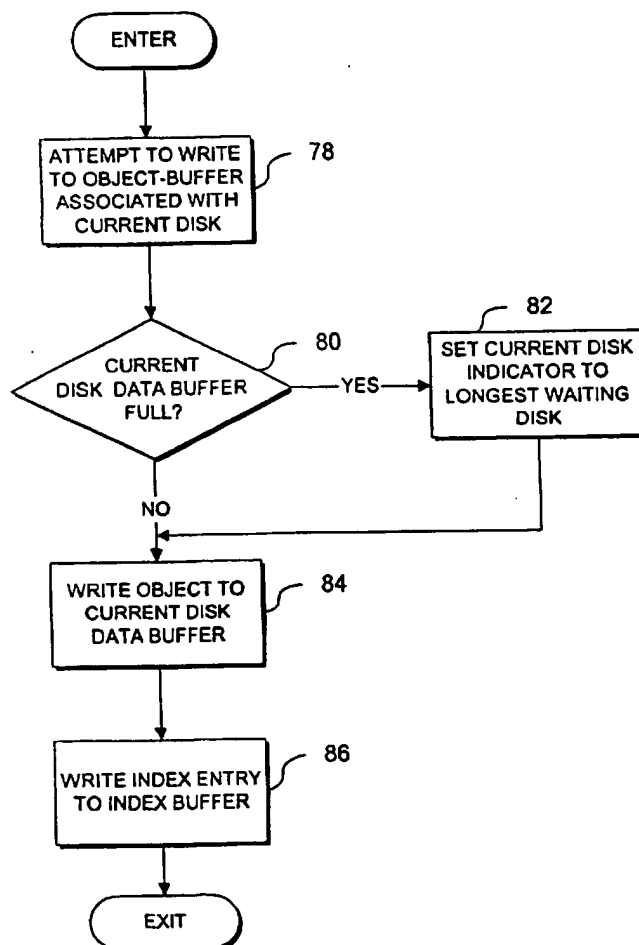
JAMES E. ROGAN  
*Director of the United States Patent and Trademark Office*



US 20040111443A1

(19) **United States**(12) **Patent Application Publication****Wong et al.**(10) **Pub. No.: US 2004/0111443 A1**(43) **Pub. Date: Jun. 10, 2004**(54) **APPARATUS AND METHODS FOR  
CACHING OBJECTS USING MAIN  
MEMORY AND PERSISTENT MEMORY****Publication Classification**(51) **Int. Cl.<sup>7</sup> ..... G06F 12/00**(52) **U.S. Cl. .... 707/202; 711/203; 714/2;  
711/118; 711/133**(76) **Inventors: Thomas K. Wong, Pleasanton, CA  
(US); Panagiotis Tsigotis, Mountain  
View, CA (US); Sanjay R. Radia,  
Fremont, CA (US); Rajeev Chawla,  
Union City, CA (US); Omid  
Ahmadian, Menlo Park, CA (US)****Correspondence Address:  
Finnegan, Henderson, Farabow,  
Garrett & Dunner, L.L.P.  
1300 I Street, N.W.  
Washington, DC 20005-3315 (US)**(57) **ABSTRACT**

An object cache stores objects in a cyclic buffer to provide highly efficient creation of cache entries. The cache efficiently manages storage of a large number of small objects because the cache does not write objects into a file system as individual files, rather the cache utilizes cyclical buffers in which to store objects as they are added to the cache. Because of the use of a cyclic buffer, the high-overhead process of purging cache entries never needs to be performed. Cache entries are automatically purged as they are overwritten when the cyclic buffer becomes full and the input pointer wraps around from the end of a cyclic buffer to the beginning of a cyclic buffer. Additionally, in the event of a system crash or disk subsystem malfunction, inspect and repair time is independent of the size of the cache, as opposed to conventional file systems in which the time is proportional to the size of the file system.

(21) **Appl. No.: 10/137,443**(22) **Filed: May 3, 2002****Related U.S. Application Data**(60) **Provisional application No. 60/304,173, filed on May  
30, 2001.**

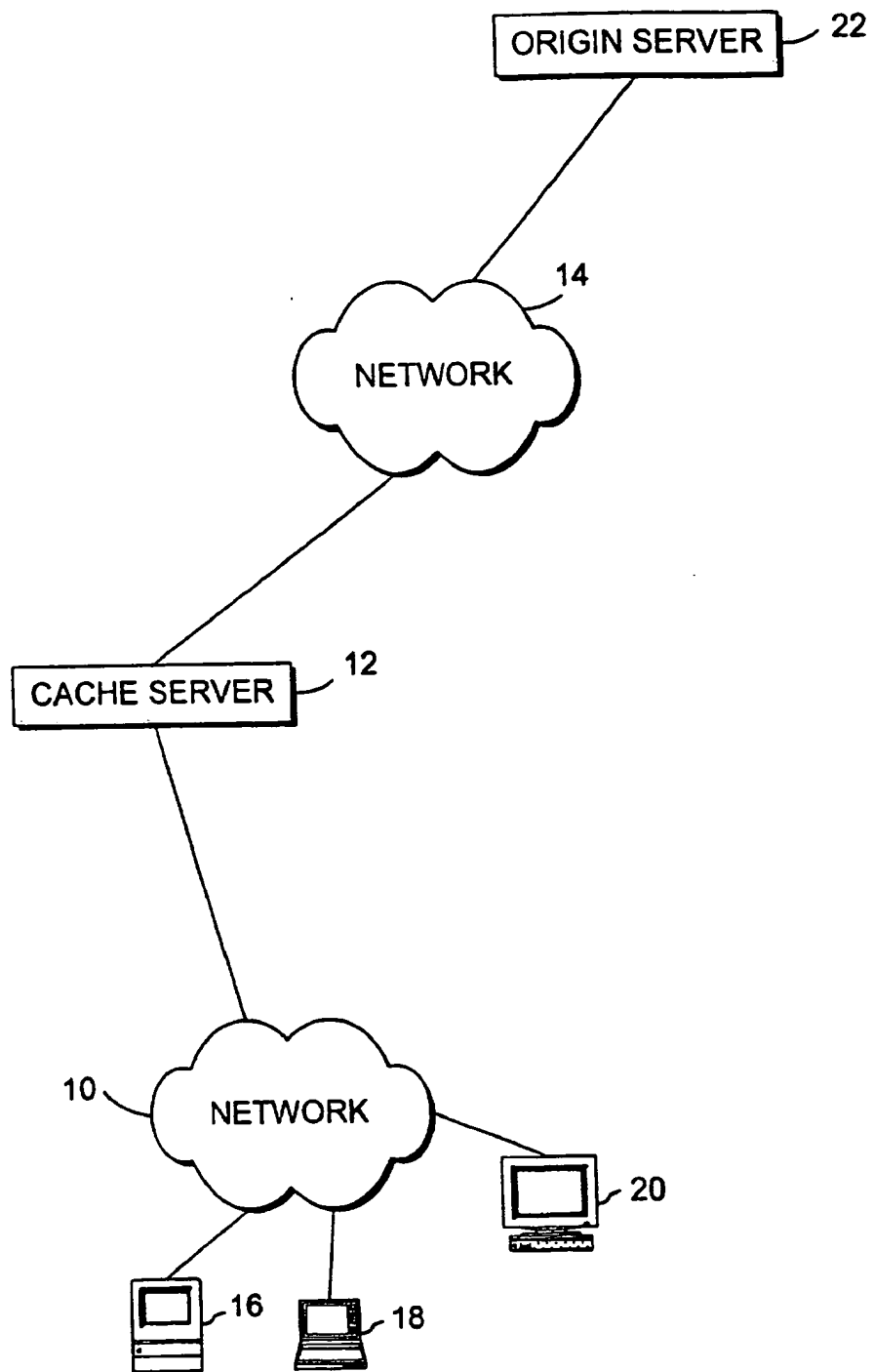


Fig. 1

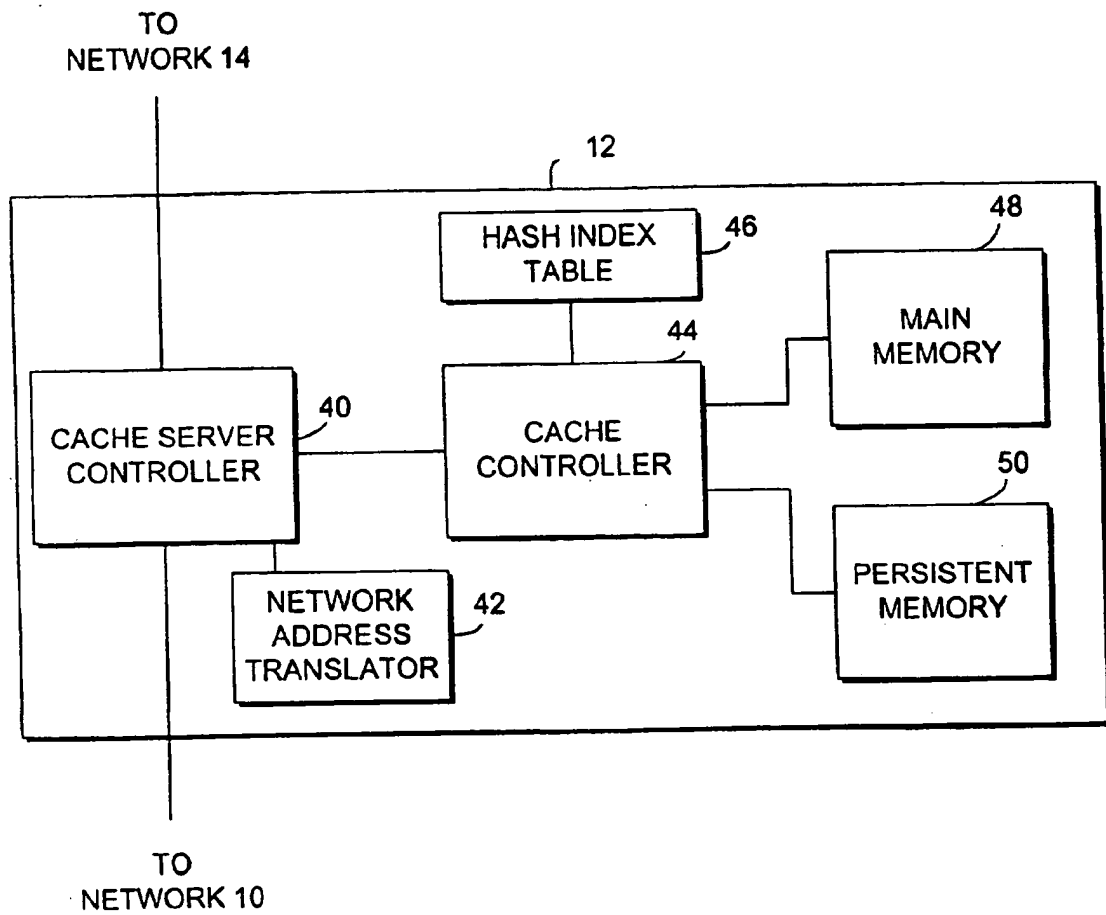


Fig. 2

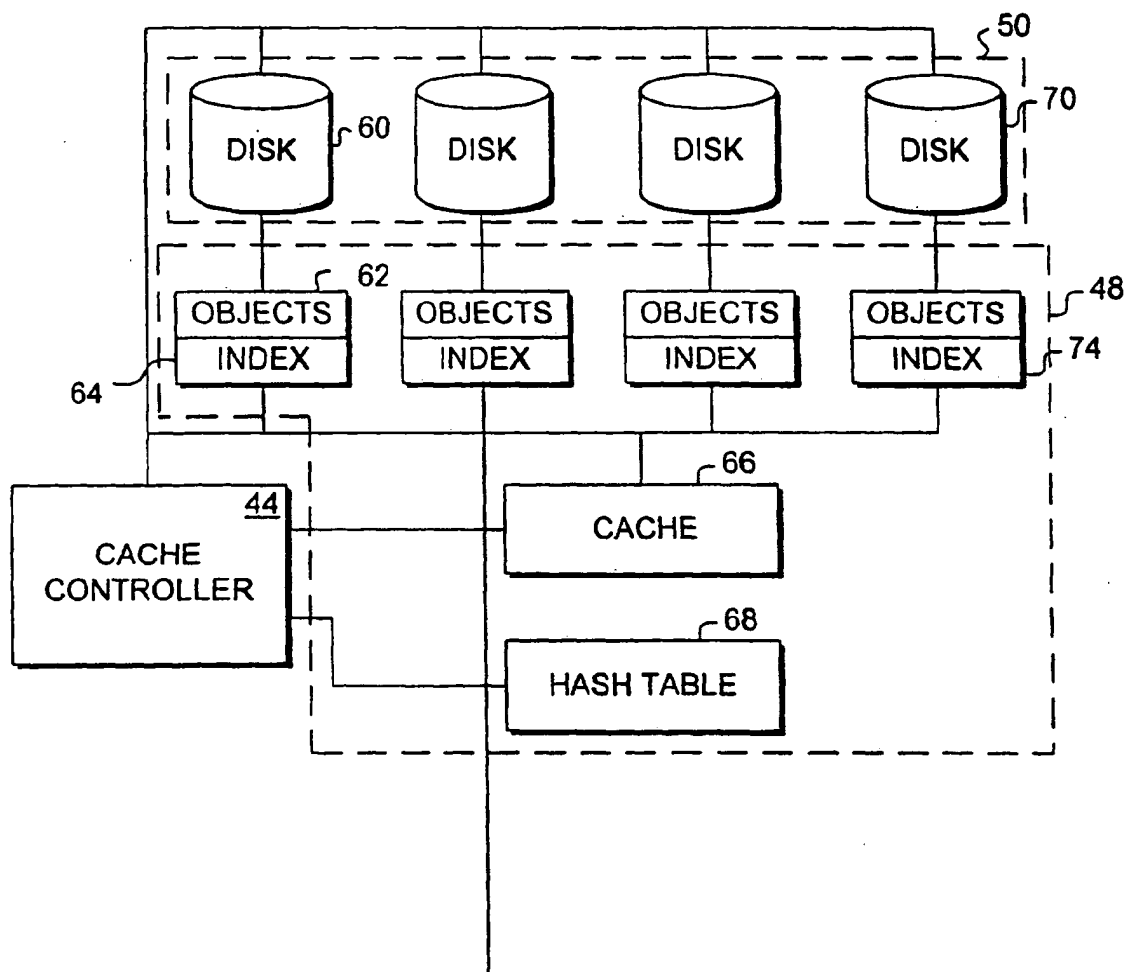


Fig. 3

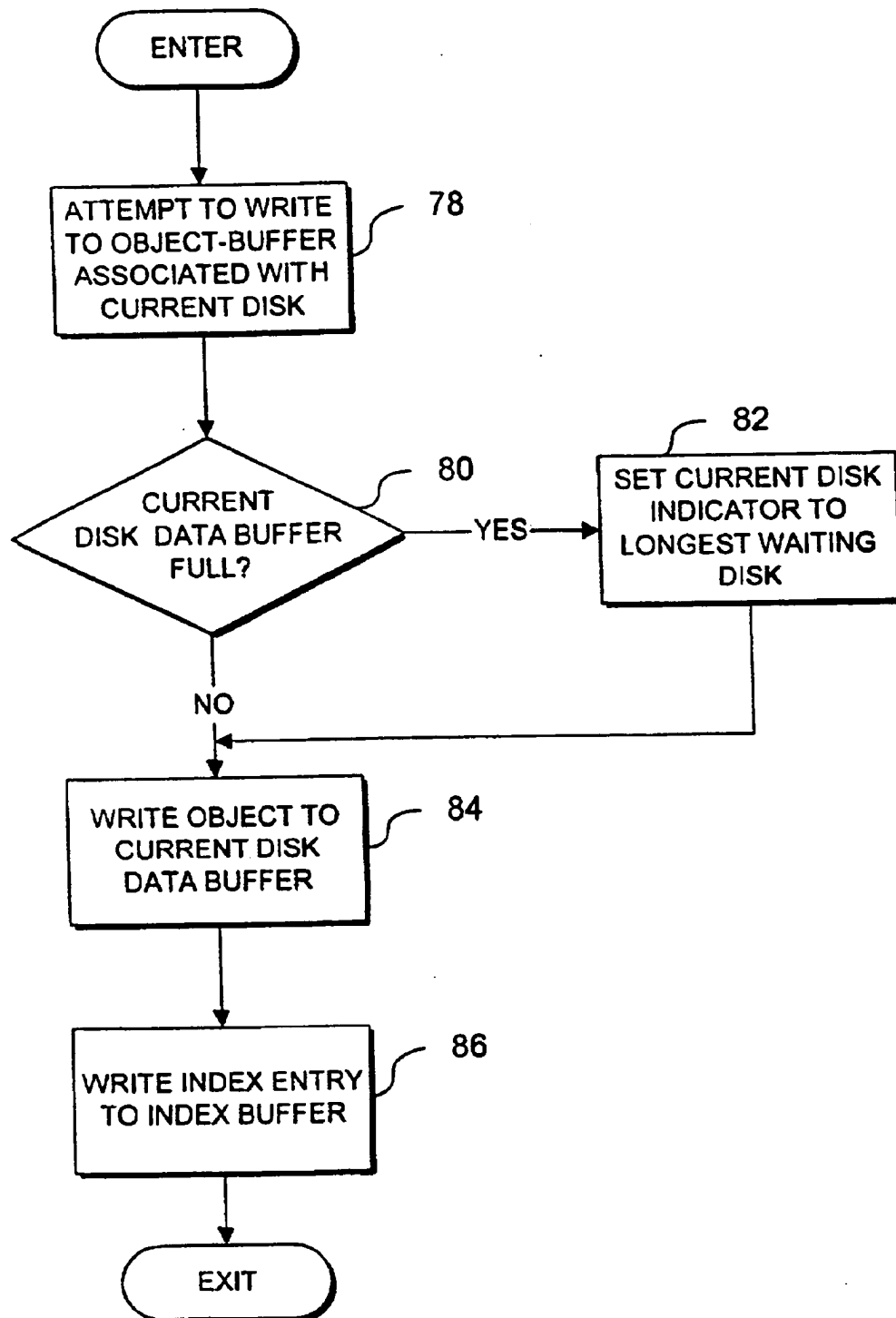


Fig. 4

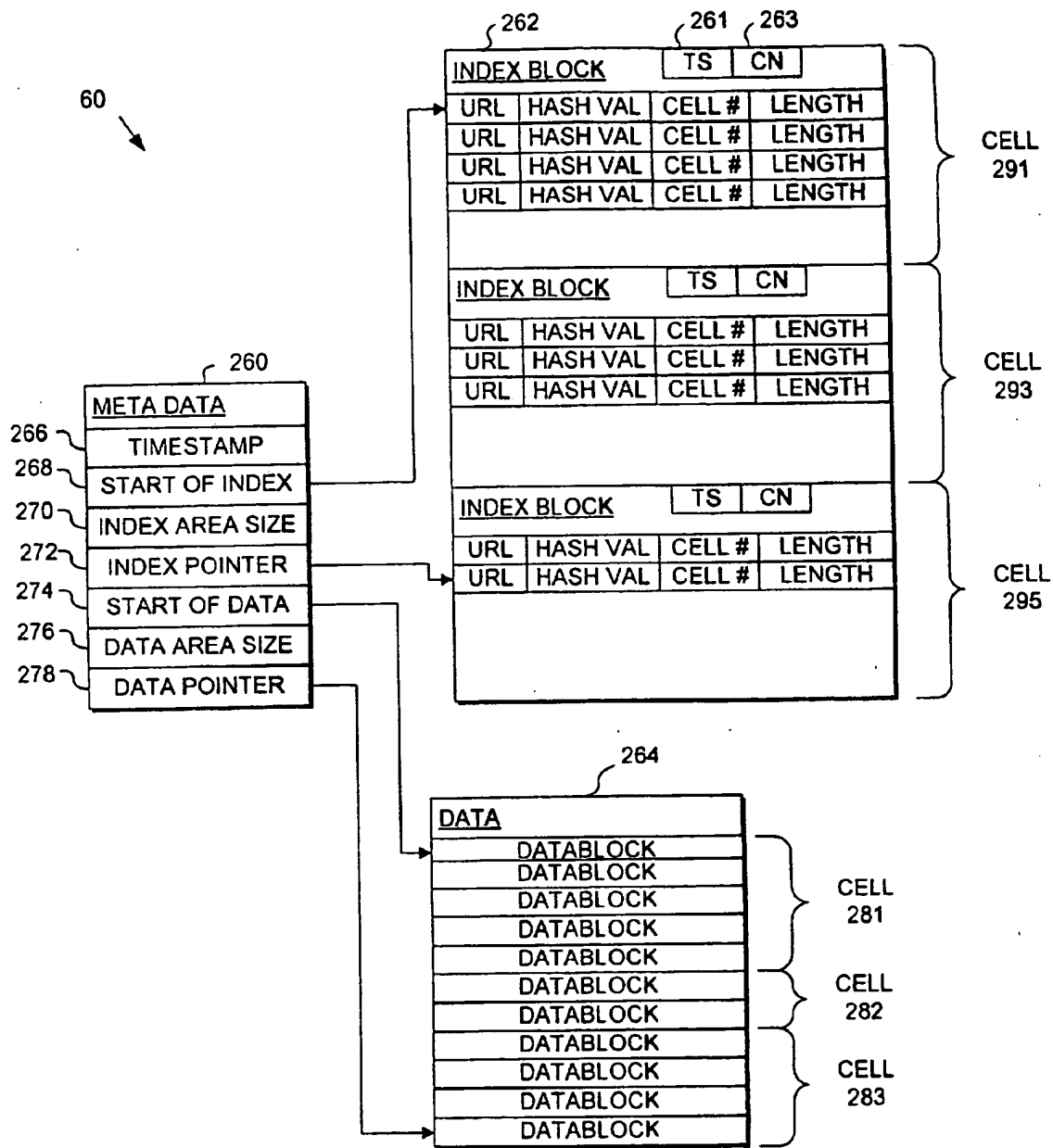


Fig. 5

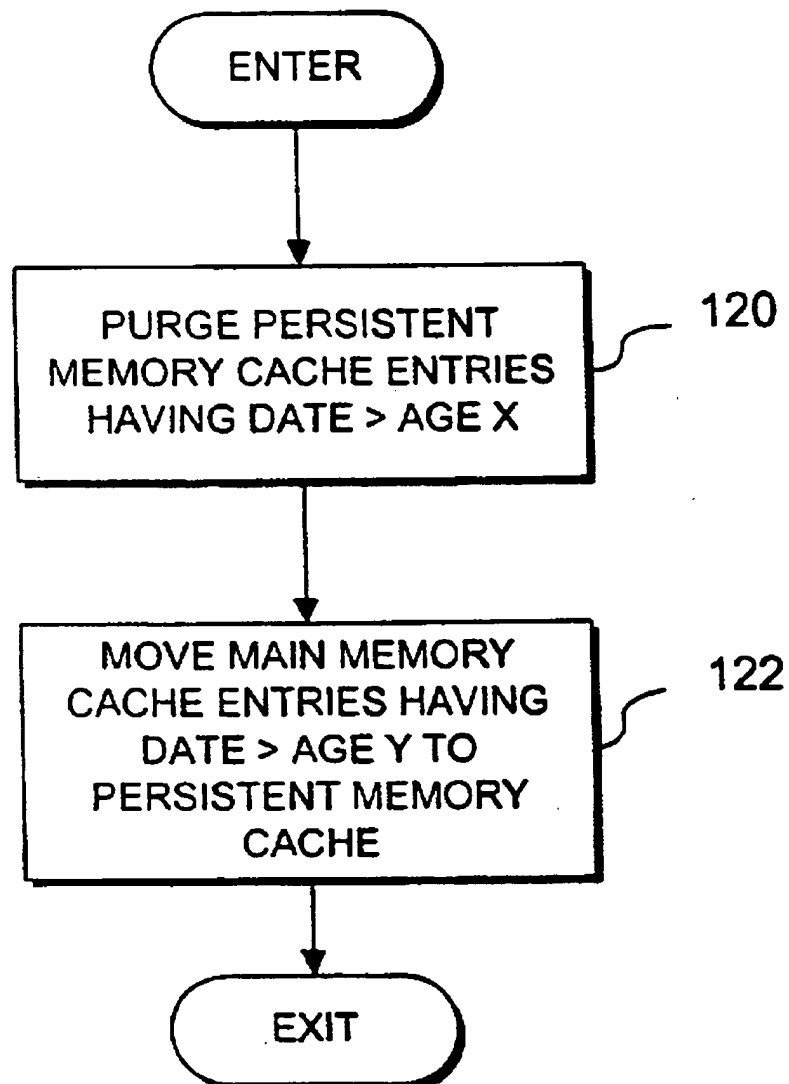


Fig. 6

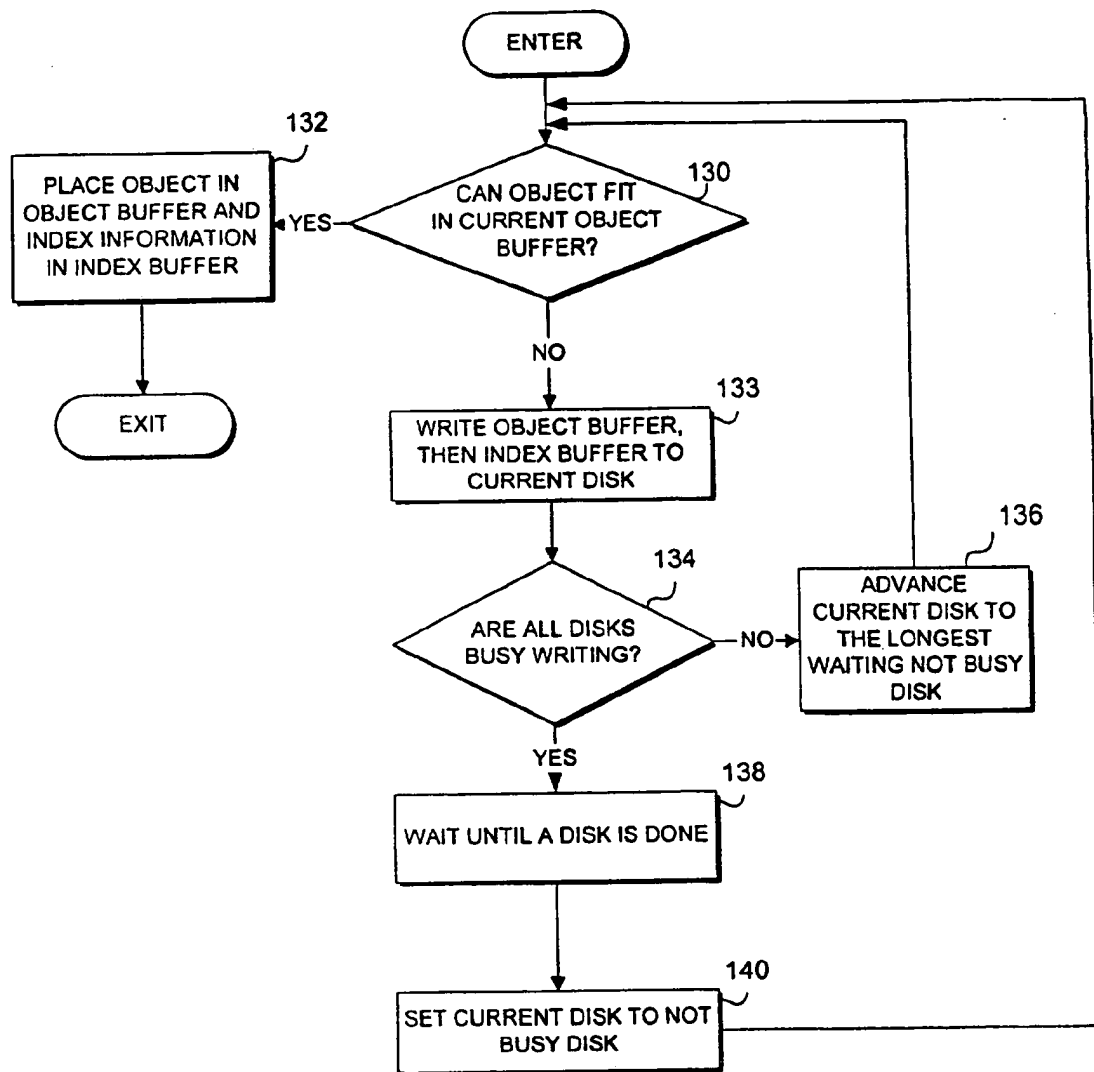


Fig. 7

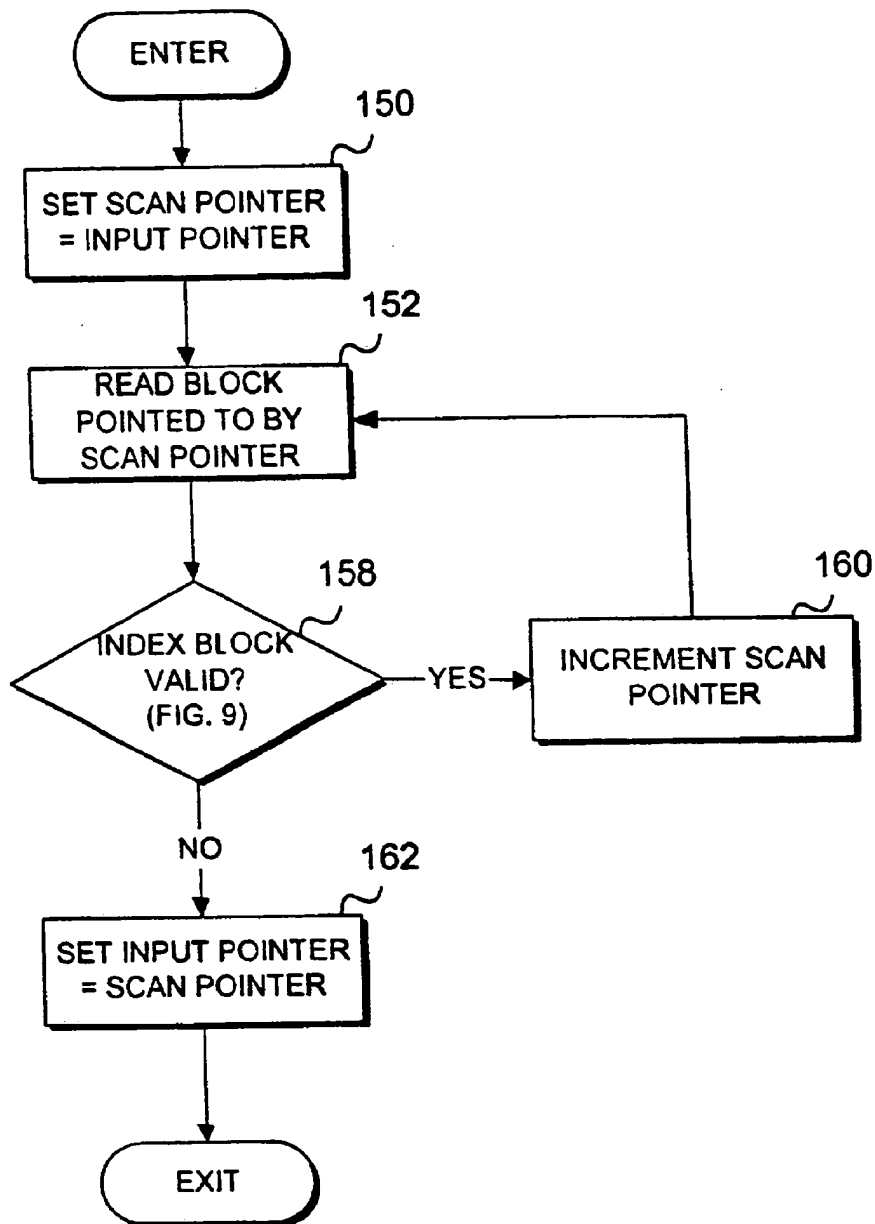


Fig. 8

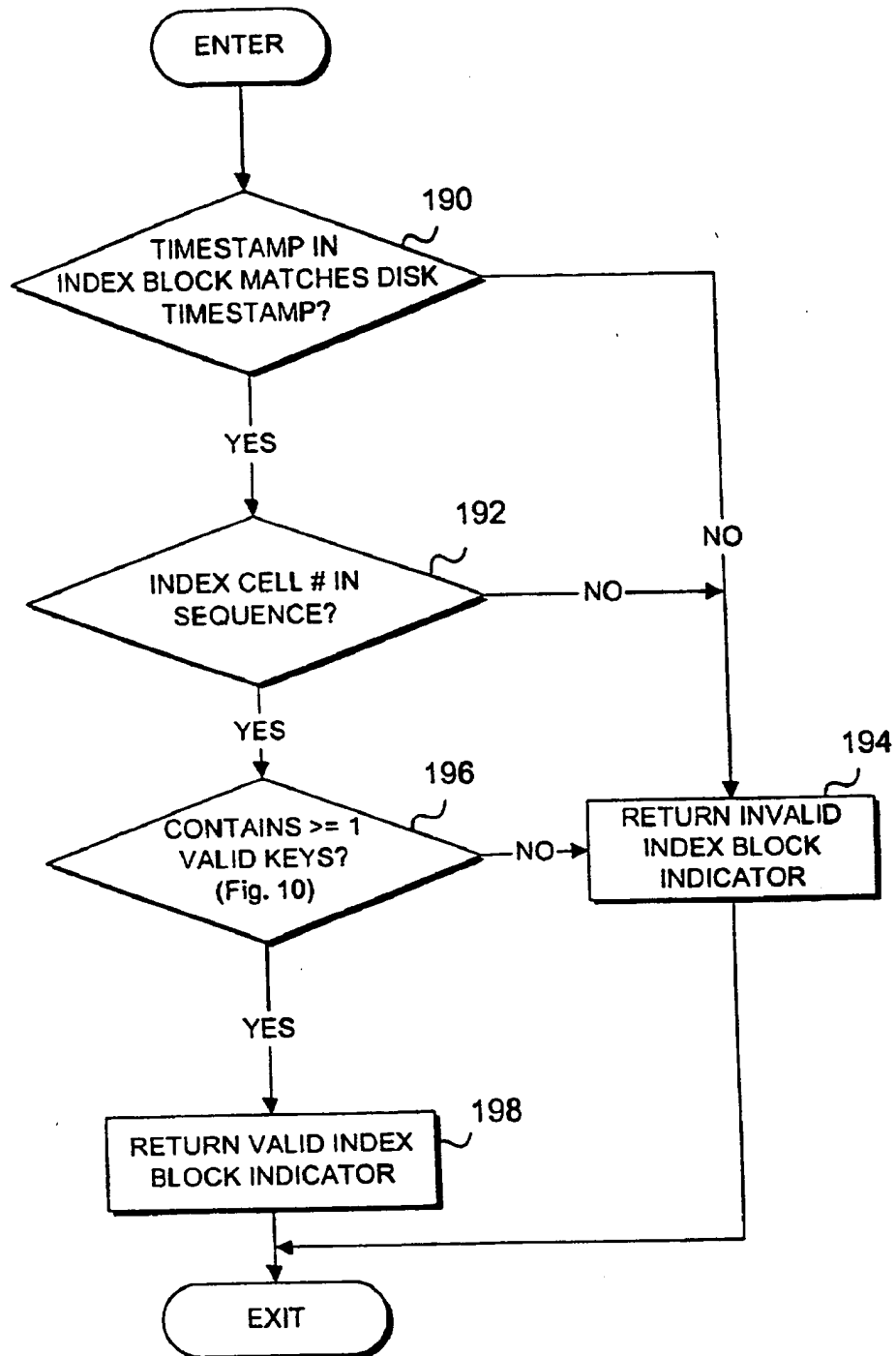


Fig. 9

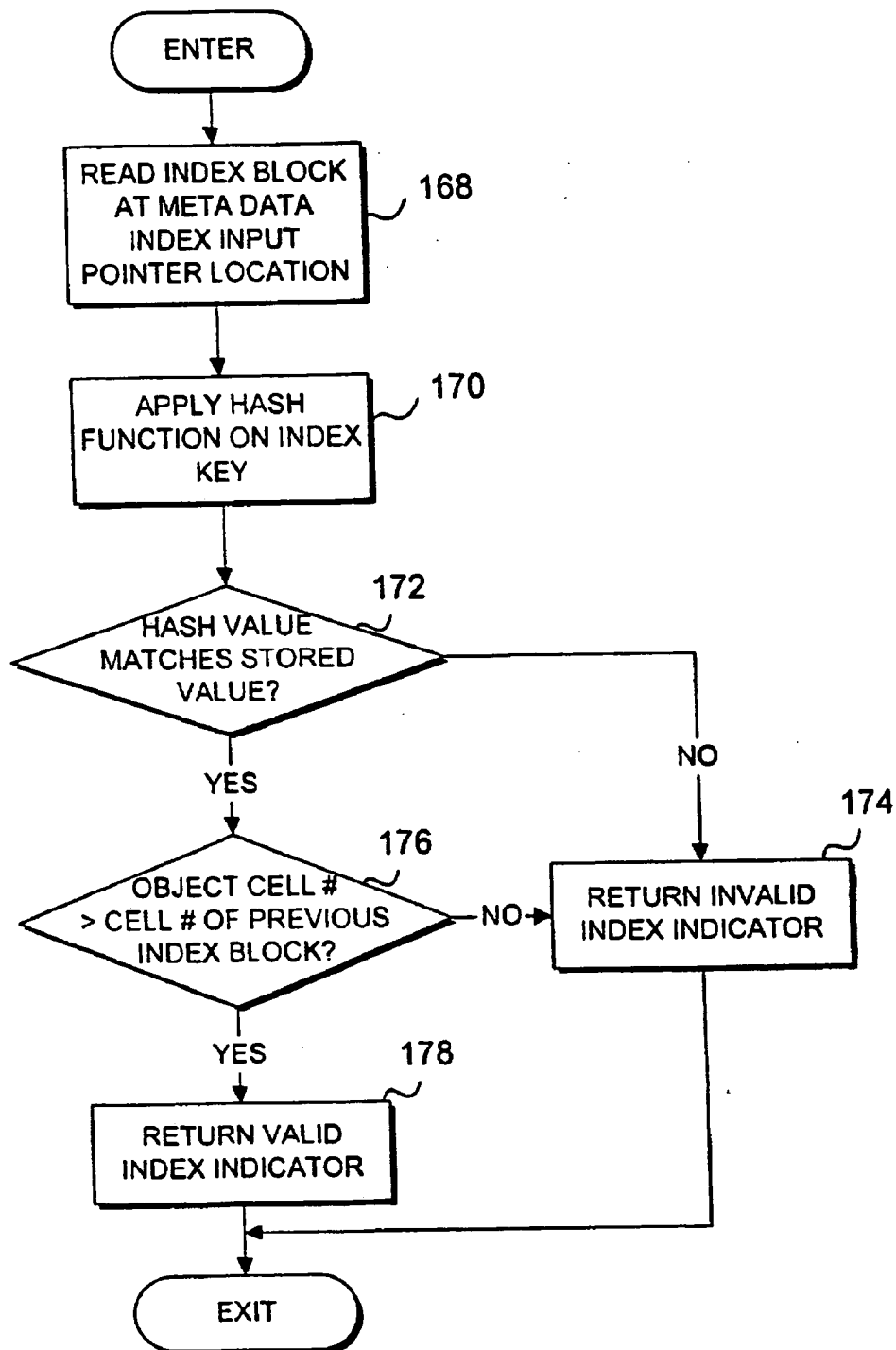


Fig. 10

## APPARATUS AND METHODS FOR CACHING OBJECTS USING MAIN MEMORY AND PERSISTENT MEMORY

### BENEFIT OF PRIORITY

[0001] The present application claims benefit of priority of U.S. Provisional Patent Application No. 60/304,173 filed on May 30, 2001.

### RELATED APPLICATION

[0002] The present invention is related to U.S. patent application Ser. No. 09/288,023, filed Apr. 8, 1999, which is incorporated herein by reference in its entirety.

### DESCRIPTION OF THE INVENTION

#### FIELD OF THE INVENTION

[0003] The present invention relates generally to caching objects in a computer system and in particular to an apparatus and method for caching objects using main memory and persistent memory to increase caching performance of persistent memory and improve recovery from file system crashes.

#### BACKGROUND OF THE INVENTION

[0004] The Internet is a global network of inter-connected computer systems that is widely used to access a vast array of information. A web server is an Internet-connected computer system that can provide information to Internet-connected web client computers. The provided information is accessed in units that are sometimes called objects. The objects may be web objects, for example. Web clients connect to web servers and access web objects. The primary web server from which a web object originates may be called an origin web server or simply an origin server.

[0005] As Internet access becomes more popular, less expensive and faster, and as the number of web clients increases, so does the number of connections made to particular origin servers. This increased number of connections can increase both network load and server load, sometimes causing parts of the Internet and particular origin servers to become so overloaded that they provide poor responses or even become inaccessible. Web cache systems can be used to reduce both network load and origin server load by migrating copies of popular documents from origin servers to cache servers. Web cache systems also provide faster responses to client computer systems. Cache servers may reside on a network, in locations close to particular web clients. Closeness or proximity in a networking context generally refers to the number of intermediate network segments data must traverse to move from client to server. By locating a caching system "closer" to clients, data may take a shorter path between the caching system and the client than the path necessary to transmit data from a corresponding origin server to the client. Therefore, placing caching systems closer to clients generally improves response times and decreases network load.

[0006] Various means for caching data are often used in computer systems to reduce system loads by storing copies of frequently accessed information in places closer to where the information is likely to be needed. In response to a request from an information requester, a cache system

determines whether the information is cached. If not, it is called a cache miss, and the cache system requests the information from the appropriate information provider and caches a copy of the information. If the information is already in the cache, it is called a cache hit. In either case, the information is then forwarded to the requester. By thus reducing loads and locating data closer to where it is needed, caching systems make data access faster and more efficient.

[0007] Cache systems can therefore improve the speed of information transfer between an information requester and an information provider. Cache systems generally include a high-speed memory that stores information frequently requested or most recently requested from an information provider by an information requester. A cache is often used in networks to expedite transferring information between a client, the information requester, and an origin server, the information provider.

[0008] A protocol defines how the client and origin server communicate. The Internet, for example, uses protocols such as the Hypertext Transfer Protocol ("HTTP") to transfer web objects. In HTTP, a client sends a service request message to the origin server. For example, the service request message might include a request method to be performed on an object in the origin server. The object is identified in the service request message by a uniform resource locator ("URL"), which is a path to the object. The origin server responds to the service request message by performing the method in the request method. The request method may be a retrieve operation, which causes the origin server to retrieve the object identified by the URL and transmit it to the requesting client.

[0009] A proxy server is often used as an intermediary between a client and an origin server. Instead of the service request message being sent to the origin server, it may be routed to a proxy server. The proxy server handles the request by forwarding it to the origin server, receiving the requested information from the origin server, and transmitting the requested information to the client.

[0010] To expedite retrieval of the requested information, a conventional proxy server often also acts as a cache, storing retrieved information in a local cache in the proxy server. When a caching proxy server receives a service request message from a client, it first determines whether the information requested by the service request message is already stored locally in the proxy server cache. If the requested information is stored in the proxy server cache, it is retrieved from the cache and returned to the client. If the information is not stored in the proxy server cache, the proxy server requests the information from the origin server. When the caching proxy server receives the requested information from the origin server, it sends the information to the client and also caches the information in the local proxy server cache. Caching can alternatively or additionally be done at other locations in the system. For example, a web browser can cache information locally at the client.

[0011] Some time after an object is cached, it may be modified on the origin server. Therefore, if a client obtains an object from a cache server instead of the origin server, the client bears some risk that the cached object may not correspond to the most up-to-date web object on the origin server. When a cached object becomes out-of-date, it is sometimes referred to as being stale. Additionally, the longer

an object remains in cache, the greater potential likelihood that it will be stale. In order to mitigate the problem of stale objects, web caches frequently maintain expiration dates in conjunction with objects. When a web client requests a web object that is expired, a web cache system will fetch a new copy of the expired web object from the corresponding origin server, replacing the expired object in the cache with the newly fetched object. Expired objects that have not been accessed for a predetermined time are periodically purged from the cache.

[0012] Ideally, a cache would be implemented entirely in fast main memory. In general, however, this is not practical because implementing a sufficiently large cache with only main memory would be prohibitively expensive. Moreover, when a computer crashes, such as when power goes down, fast main memory loses all of its data.

[0013] One solution to this problem is to implement a first part of a cache in main memory, and a second part of the cache in persistent memory, such as disk storage. Because persistent memory does not lose data when a computer system crashes, at least part of the cache is retained upon power loss, even though the data in main memory cache is lost.

[0014] Persistent memory has drawbacks. One problem with persistent memory is that it is slow. If there are large numbers of relatively small objects being written to and read from a traditional file system stored in persistent memory, a system can be unacceptably slow. For example, typical HTML documents may be as small as 10K, or even smaller, and there may be many such objects that need to be cached. Using a traditional file system for storing individual cached objects as small files is generally inefficient for several reasons.

[0015] Internet or web caching systems ordinarily experience hit-rates of approximately 30%. This means that roughly 70% percent of the time a caching system will experience a cache miss, meaning a requested object is not found in the cache. When a cache miss is experienced, an old cache entry may be deleted to make room for an object corresponding to the object requested in the cache miss. In a traditional file system, deleting an old entry and adding a new entry involves writing to directory structures, file descriptors, and blocks containing the data corresponding to the content of the object itself. This process results in a relatively large number of writes, all potentially in different physical locations of the persistent storage medium. Traditional file systems are not well suited to this type of writing of small pieces of information.

[0016] Traditional file systems are not designed to run at 100% capacity, while a cache will ideally make full use of whatever storage capacity is available to it. When a traditional file system becomes nearly completely full, performance degrades and reads and writes are slower than on a traditional file system having some free space. This is particularly true when the traditional file system is full of small files.

[0017] When a traditional file system is used to store a multitude of small files in a relatively flat directory structure, directory traversal must be performed each time a file is read and written. Having many large files leads to having large directory lists, and traversing large directory lists in a

traditional file system is costly in terms of time. The ability of a cache system to perform rapidly is degraded by slow directory traversal in a traditional file system.

[0018] The Berkeley UNIX file system ("ufs") is one example of a system sometimes used to implement cache on disk. More particularly, ufs uses synchronous disk input and output ("I/O") operations during file creation and deletion to update a directory containing files to be created or deleted. A cache using ufs to store web objects as individual files suffers from a drastic drop in throughput due to idle time spent waiting for the disk I/O completions when creating or removing individual files containing web objects.

[0019] Another problem with traditional file systems involves recovery after a file system crash. To recover from a file system crash, the proxy server performs a file system check ("fsck") to recreate the file system prior to the crash to the extent possible. With large persistent memory, however, fsck may take a long time, because the process of repairing a traditional file system is proportional to its size. Because it takes so long to repair a large file system containing cached objects as individual files, a common solution is to simply reinitialize the file system. However, reinitialization has the drawback of losing all of the cached documents, which temporarily eliminates the benefits of a cache, until the cache begins to accumulate new objects that are added to the cache as a result of cache misses.

[0020] What is needed, then, is a system which allows high performance use of persistent memory and provides quick recovery in the event of a file system crash without requiring the loss of all data.

#### SUMMARY OF THE INVENTION

[0021] Methods and systems, consistent with the present invention, provide an object cache by using cyclic buffers in conjunction with fast main memory and persistent memory. First, the systems and methods receive an object from an origin server. Next, they store the object in a plurality of data structures comprising three elements. The first element is a cyclic index buffer capable of storing index entries comprising an index. The second element is a cyclic object buffer capable of storing and retrieving objects by logical block number. The third element is a metadata buffer capable of storing information about the index buffer and the object buffer.

[0022] Methods and systems consistent with the present invention are applicable to caching web content, data files, binary executable files, and other types of information. Although a web caching system is provided as an exemplary embodiment, a person of ordinary skill will appreciate that the present invention may be practiced in connection with other types of caching systems without departing from the scope of the present invention as claimed.

[0023] Additional benefits of the present invention will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practice of the invention. The benefits of the present invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims.

[0024] It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention, as claimed.

## BRIEF DESCRIPTION OF THE DRAWINGS

[0025] The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate exemplary embodiments of the invention and together with the description, serve to explain the principles of the present invention. In the drawings,

[0026] FIG. 1 is a block diagram showing a system implementing a proxy server consistent with the present invention;

[0027] FIG. 2 is a block diagram showing proxy server 12 in greater detail;

[0028] FIG. 3 is a block diagram illustrating the main memory and persistent memory portions of a cache, consistent with the present invention;

[0029] FIG. 4 is a flow chart showing the processing performed by a cache controller when writing an object to a persistent memory cache consistent with the principles of the present invention;

[0030] FIG. 5 is a schematic diagram illustrating the contents of a disk consistent with the principles of the present invention;

[0031] FIG. 6 is a flow chart illustrating the process performed by a cache controller when purging and staging entries in accordance with the principles of the present invention;

[0032] FIG. 7 is a flow chart illustrating the process performed by a cache controller when writing an object to main memory consistent with the principles of the present invention;

[0033] FIG. 8 is a block diagram illustrating the process performed by a cache controller in determining the location of valid information in persistent memory after a system crash consistent with the principles of the present invention;

[0034] FIG. 9 is a flow chart illustrating the process performed by a cache controller when determining whether an index block is valid consistent with the principles of the present invention; and

[0035] FIG. 10 is a flow chart illustrating the process performed by a cache controller in determining whether an index entry in an index block is valid consistent with the principles of the present invention.

## DETAILED DESCRIPTION

[0036] Reference will now be made in detail to the present exemplary embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

[0037] Systems, methods, and articles of manufacture consistent with the caching scheme disclosed and claimed herein provide the advantages of main memory and persistent memory caching with improved performance and failure recovery as compared to conventional systems. Systems and methods consistent with the improved cyclic buffer disclosed herein use an index to map consecutively numbered logical blocks to physical buffer blocks of a cyclic buffer. Cyclic buffers consistent with the present invention are disclosed in related U.S. patent application Ser. No.

09/288,023, filed Apr. 8, 1999, which is incorporated herein by reference in its entirety. The index to the cyclic buffer is implemented using the logical block numbers. When new information is added to the cyclic buffer, the next logical block number is assigned to the information. The new logical block number corresponds to a physical cyclic buffer block. The new logical block number and key associated with the information are then added to the index. The mapping of logical block numbers to physical block numbers may simply be a mathematical formula that defines the relationship between the logical block numbers and physical block numbers.

[0038] To retrieve information from the cyclic buffer, the index is first scanned to determine whether an index entry exists that has a key associated with the information. If such a key is present, the entry is accessed to determine the logical block number associated with the key. The logical block number is then mapped to a physical block number, and the physical block number is used to access the cyclic buffer.

[0039] In one implementation, a logical block number C is converted into a physical block number by determining a remainder after dividing the block number C by the number of blocks N in the buffer (i.e.,  $C/N$ ). The logical block number can be used to determine which blocks are the newest, as well as whether the block is valid. The newest block is the block having the highest logical block number. The block is a valid block if the logical block number being requested is within the range of logical block numbers currently being used to implement the buffer. Thus, logical block numbers are used to both identify the blocks within a cyclic buffer and to determine whether a block has been overwritten (i.e., is invalid).

[0040] In one implementation of a cyclic buffer consistent with the invention, a caching proxy server retrieves information from a network and stores the information in the cyclic buffer using a logical block buffer control mechanism. This allows the information to be accessed at a later time without retrieving it from the network.

[0041] FIG. 1 is a block diagram showing a system implementing a cache server consistent with the present invention. Clients 16, 18, and 20 are connected to network 10. Cache server 12, connected between network 10 and network 14, handles service requests from clients 16, 18, and 20. In response to a service request for information from one of the clients, such as client 16, cache server 12 either returns the information from its local cache, or retrieves the information from origin server 22 and forwards the information to client 16. If the information is retrieved from origin server 22, cache server 12 also caches the information locally for later use. Caching the information allows cache server 12 to quickly provide the information to a client at a later time if the same information is requested again.

[0042] The apparatus and methods consistent with the invention are related to cache servers and proxy server caching. A cache server consistent with the invention may be implemented in whole or in part by one or more sequences of instructions, executed by cache server 12, which carry out the apparatus and methods described herein. Such instructions may be read by cache server 12 from a computer-readable medium, such as a storage device. Execution of sequences of instructions by cache server 12 causes perfor-

mance of process steps consistent with the present invention described herein. Execution of sequences of instructions by cache server 12 may also be considered to implement apparatus elements that perform the process steps. Hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0043] The term "computer-readable medium" as used herein refers to any medium that may store instructions for execution. Such a medium may take many forms, including but not limited to, non-volatile memory media, volatile memory media, and transmission media. Non-volatile memory media includes, for example, optical or magnetic disks. Volatile memory media includes RAM. Transmission media includes, for example, coaxial cables, copper wire and fiber optics, including the wires. Transmission media can also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

[0044] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic storage medium, a CD-ROM, DVD, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, an EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read and use.

[0045] Various forms of computer-readable media may be involved in carrying one or more sequences of instructions for execution to implement all or part of the cyclic cache described herein. For example, the instructions may initially be carried on a magnetic disk or a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to a computer system can receive the data on the telephone line and use an infrared transmitter to convert the data to an infrared signal. An infrared detector coupled to appropriate circuitry can receive the data carried in the infrared signal and place the data on a bus. The bus may carry data to a memory, from which a processor retrieves and executes the instructions. The instructions received by the memory may optionally be stored on a storage device either before or after execution by the processor.

[0046] FIG. 2 is a block diagram showing cache server 12 in greater detail. Cache server 12 may be implemented by programming a conventional computer, as is well understood in the art. The blocks shown in FIG. 2 may be implemented in hardware, software, or a combination of hardware and software.

[0047] Cache server 12 is controlled by cache server controller 40. Cache server controller 40 is connected to a network address translator ("NAT") 42 and a cache controller 44. Cache controller 44 is connected to a hash index table 46, a main memory 48, and a persistent memory 50. Hash index table 46 contains an index of the information cached in main memory 48 and persistent memory 50. Cache server controller 40 receives requests for information from clients 16, 18, and 20, through network 10. For example, client 16 sends a service request message requesting information from

origin server 22. Cache server controller 40 receives the service request message and sends the request to cache controller 44. Cache controller 44 accesses information in hash index table 46 to determine whether the requested information is present in main memory 48 or persistent memory 50. If the information is present in main memory 48 or persistent memory 50, cache controller 44 retrieves the information and returns it to cache server controller 40. Cache server controller 40 then forwards the information to client 16, via network 10.

[0048] If cache controller 44 determines from hash index table 46 that main memory 48 and persistent memory 50 do not contain the requested information, cache controller 44 sends an indication to cache server controller 40 that the information is not cached. Cache server controller 40 responds by sending a service request message to origin server 22 to request the requested information. Upon receiving the requested information from origin server 22, cache server controller 40 sends the information to client 16. Cache server controller 40 also forwards the information to cache controller 44, which stores the information in main memory 48 and updates hash index table 46 with information defining the location of the new information in main memory 48.

[0049] FIG. 3 is a block diagram illustrating a system for implementing main memory 48 and persistent memory 50 consistent with the principles of the invention. In one embodiment, persistent memory 50 is comprised of one or more disks 60, 70. Main memory 48 includes object buffers and indexes, such as object buffer 62 and index buffer 64, corresponding to each of the disks in persistent memory 50. Main memory 48 also includes cache 66 and hash table 68. Hash table 68 stores information for accessing entries in cache 66. Cache controller 44 stores information in the disks of persistent memory 50 by staging the information from cache 66 to the index and object buffers of main memory 48, and then writing the buffered information to persistent memory 50.

[0050] Buffering the objects and associated index information allows the disks of persistent memory 50 to perform more efficiently because writes can be grouped, thus reducing the overhead of multiple write operations spread across time. This is because many disks require the same amount of time to write a small amount of data as they do to write a larger amount of data and therefore by grouping smaller chunks of data into a larger chunk, more data can be written per unit of time. Although accumulating objects in main memory and grouping writes helps performance, should the system go down, information in the index and object buffers could be lost. To recover, the system must determine where the valid information is located in persistent memory 50. Methods and apparatus consistent with the invention facilitate recovery by writing information to main memory 48 and persistent memory 50 in a particular sequence. The particular sequence of writing operations allows the system to determine the location of valid information in persistent memory 50 more quickly than conventional systems.

[0051] FIG. 4 is a flowchart showing the process performed by cache controller 44 when writing an object to a disk in persistent memory 50. FIG. 4 is described with reference to FIG. 3. In this embodiment, cache controller 44, of FIG. 3, only writes to one disk at a time. The disk

currently being written to is referred to as the current disk. Assuming for purposes of discussion that disk 60 is the current disk, cache controller 44, of FIG. 3, first attempts to write the object to corresponding object buffer 62 of the current disk, and corresponding index information related to the object to index buffer 64 (step 78). If object buffer 62 is full (step 80), cache controller 44 sets a current disk indicator to the disk that has been waiting the longest, so that information will be written to the longest waiting disk (step 82). Assuming for the sake of discussion that disk 70 in FIG. 3 has been waiting the longest, disk 70 becomes the current disk. The object is written to the object buffer (step 84) and its index information to its associated index buffer (step 86) of the current disk. For example, if there are only two disks, cache controller 44 writes the object and index information to the object buffer and index buffer associated with the second disk if the object buffer associated with the first disk is full.

[0052] The information in each object buffer and index buffer of main memory 48 is periodically written to the respective disks of persistent memory 50. If more than one object buffer is waiting to be written to disk, the object buffer waiting the longest is written first. Similarly, if more than one index buffer is waiting to be written, the index buffer that has been waiting the longest is written first. An index buffer will not be written unless the associated object buffer has already been written. Buffering the objects and index information allows the system to perform writes to persistent memory 50 in one operation. This reduces the performance penalty normally caused in conventional systems by frequent and sporadic writes to persistent memory 50, because in many persistent memory systems, such as disk drives, there is a constant access time associated with writing a discrete block of data whether the entire block is written or only part of that block. Therefore, it can be much more efficient to accumulate data before performing a write operation.

[0053] FIG. 5 is a schematic diagram illustrating memory areas of a disk, for example, disk 60. Disk 60 includes three memory areas: meta data 260, index 262, and data 264. Metadata is data about data, and it contains information about how data is stored and formatted on the disk. Metadata 260 is stored at the same location on each disk, and stores information for accessing index 262 and data 264. Metadata includes a timestamp 266, a start of index pointer 268, an index area size indicator 270, and an index pointer 272, a start of data pointer 274, a data area size indicator 276 and a data pointer 278. Timestamp 266 is stored in metadata 260 at the time disk 60 is initialized. Start of index 268 stores the location where index 262 starts, index area size indicator 270 indicate the size of index 262, and index pointer 272 points to the most recently written index information in index 262. Start of data pointer 274 stores the location where data 264 begins, data area size indicator 276 indicates the size of data 264, and data pointer 278 points to the most recently written data in data 264.

[0054] Data 264 stores objects and may be comprised of data blocks of equal size, corresponding to the data block size of the persistent storage device. Objects are stored in groups of one or more data blocks. Each group is called a cell. In FIG. 5, exemplary cells 281, 282, and 283 are illustrated. In one embodiment, data 264 is implemented as

a cyclic data structure of cells such that the operation of advancing a pointer past the last cell causes the pointer to point to the first cell.

[0055] Index 262 comprises index blocks. Each index block is stored in a respective one of cells 291, 293, or 295. At the time each index block is created, timestamp (TS) 261 and cell number (CN) 263 are written to the index block. Each index entry stores particular information regarding a data cell stored in data 264. More particularly, each index entry stores the URL of the object stored in a data cell, a hash value of the URL, the data cell number, and the data cell size.

[0056] Index 262 is only used at startup time to create a hash table in fast main memory. The hash table is used by cache controller 44 to determine the locations of web objects in data 264. In one embodiment, index 262 is implemented as a cyclic data structure of index entries.

[0057] Cache controller 44 updates index pointer 272 and data pointer 278 in metadata 260 periodically, but not each time new information is written to index 262 or data 264. The updates are only done periodically because the overhead of performing updates every time index area 262 or data area 264 are updated would be too high and would degrade performance.

[0058] Updating index pointer 272 and data pointer 278 only periodically, however, creates problems when the system crashes. For example, if the system crashes after either index 262 or data 264 are updated, but before index pointer 272 or data pointer 278 are correspondingly updated, the pointers will point to areas in each of index 262 or data 264 that are not the most recent entries. What is known, however, is that index pointer 268 and data pointer 278 point to areas that were at one time the most recently written information in index 262 and data 264. Based on this information, cache controller 44 can find index information that at one time was valid, and begin stepping through subsequent entries in index 262 until no valid index entries are found. Of course, if the system crashes at the exact time that index pointer 268 and data pointer 270 are correct, then cache controller 44 can immediately begin using the disk again.

[0059] FIG. 6 is a flow chart illustrating the process performed by cache controller 44 when purging and staging entries in main memory 48 and persistent memory 50. When an object is buffered in main memory 48 it is given a timestamp which may include a date and a time. The timestamp allows cache controller 44 to determine how old the object is. Periodically, for example, when a certain number of writes have been buffered in main memory 48, cache controller 44 purges entries from persistent memory 50 that are older than a particular age (step 120), and moves entries buffered in main memory 48 having dates greater than a particular age to persistent memory 50 (step 122). Purging entries from persistent memory 50 frees up disk space, and moving entries from main memory 48 frees up buffer space.

[0060] FIG. 7 is a flow chart illustrating the process performed by cache controller 44 when writing an object to main memory 48. Using disk 60 of FIG. 5 as an example of the current disk, cache controller 44 first determines whether the object to be written can fit in the object buffer 62 (step 130). If the object fits in object buffer 62, the object is

written to object buffer 62 and the index information associated with the object is written to index buffer 64 (step 132).

[0061] If the object does not fit in object buffer 62, cache controller 44 writes the object buffer, then the index buffer to the current disk (step 133) and determines whether all disks are busy writing (step 134). If all disks are not busy writing, cache controller 44 makes the longest waiting not busy disk the current disk (step 136), and continues the process of determining whether the object can fit in the new current disk object buffer at step 130. If all disks are busy writing, cache controller 44 waits until a disk is done (step 138), and then sets the current disk to the not busy disk (step 140). The process then continues with cache controller 44 determining whether the object can fit into the new current disk object buffer at step 130.

[0062] FIG. 8 is a block diagram illustrating the process performed by cache controller 44 in determining the location of valid information in persistent memory 50 after a system crash. The process is performed for each disk of main memory 50. Using disk 60 of FIG. 5 as an example, cache controller 44 first reads metadata 260 from disk 60 to determine the most recent index pointer, and sets a scan pointer to the value of that index input pointer (step 150). Cache controller 44 then reads the block in index area 262 pointed to by the scan pointer (step 152), and determines whether the index block is valid (step 158). Determining whether the index block is valid is illustrated in FIG. 9. If the index block is valid, the scan pointer is incremented (step 160) and cache controller 44 continues at step 152 by reading the block pointed to by the new scan pointer. If the index block is not valid, cache controller 44 sets the input pointer equal to the scan pointer (step 162). Thus, cache controller 44 has reset the input pointer to the first valid block in index buffer 64.

[0063] FIG. 9 is a flow chart illustrating the process performed by cache controller 44 when determining whether an index block is valid (step 158 of FIG. 8). Cache controller 44 first determines whether the timestamp in the index block matches the disk timestamp (step 190). If the timestamp in the index block does not match the disk timestamp, cache controller 44 returns an invalid index block indicator (step 194).

[0064] If the timestamp in the index block does match the disk timestamp, cache controller 44 determines whether the index block cell number is in sequence (step 192). If the index block cell number is not in sequence, cache controller 44 returns an invalid index block indicator (step 194).

[0065] If the index cell number is in sequence, cache controller 44 determines whether the index block contains greater than or equal to one valid index (step 196). If the index does not contain greater than or equal to one valid index, cache controller 44 returns an invalid index block indicator (step 194). If cache controller 44 determines that the index does contain at least one valid index, then cache controller 44 returns a valid index block indicator (step 198).

[0066] FIG. 10 is a flow chart illustrating the process performed by cache controller 44 in determining whether an index entry in an index block is valid. Cache controller 44 first reads the index entry from the location identified by the index input pointer stored in metadata area 260 (step 168), applies a hash function to the index key of the index entry

(step 170), and determines whether the hash value matches the stored value (step 172). If the hash value does not match the stored value, cache controller 44 returns an invalid index indicator (step 174). If the hash value does match the stored value, then cache controller 44 determines whether the object cell number of the index entry is greater than the object cell number of the previous index entry (step 176). If the object cell number is not greater than the previous index entry, cache controller 44 returns an invalid index indicator (step 174). If the object cell number is greater than the object cell number of the previous index block, cache controller 44 returns a valid index indicator (step 178).

[0067] It will be apparent to those skilled in the art that various modifications and variations can be made in the caching system and methods consistent with the principles of the present invention without departing from the scope or spirit of the invention. Although several embodiments have been described above, other variations are possible consistent with the principles of the present invention. Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the disclosed embodiments. The specification and examples are exemplary only, and the true scope and spirit of the invention is defined by the following claims and their equivalents.

We claim:

1. A method in a computer system having main memory and persistent memory, the main memory containing at least one data buffer, and the method comprising the steps of:

receiving an object from an origin server; and

storing the object in a plurality of data structures, the data structures comprising:

a cyclic index buffer capable of storing index entries comprising an index, the index having a beginning;

a cyclic object buffer capable of storing and retrieving objects by logical block number, the object buffer having a beginning; and

a metadata buffer capable of storing information about the index buffer and the object buffer.

2. A method as in claim 1, wherein the object is a web object.

3. A method as in claim 1, further comprising the steps of:

determining whether a current disk data buffer is full, the current disk data buffer associated with a current disk; and

setting a current disk indicator to indicate a longest waiting disk if the current disk data buffer is full.

4. A method as in claim 1, wherein the step of storing objects further comprises the step of:

storing index blocks comprising:

a time stamp capable of storing a time associated with the index entry; and

a cell number capable of identifying a cell within the cyclic index buffer.

5. A method as in claim 4, wherein the step of storing index blocks further comprises the step of storing index elements including:

an object identifier capable of identifying an object;  
 a hash value corresponding to the object identifier;  
 a cell number corresponding to a storage location of the object; and  
 a cell size corresponding to the size of the cell within the index buffer.

6. A method as in claim 5, wherein the object is a web object and the object identifier is a uniform resource locator.

7. A method as in claim 1, wherein the cyclic data buffer further comprises:

cells comprising at least one data block.

8. A method as in claim 1, wherein the metadata buffer further comprises:

a timestamp capable of storing a time associated with information stored in the metadata buffer;

a start of index indicator capable of indicating a location of the beginning of the index within the index buffer;

an index buffer size capable of specifying a size of the index buffer;

an index pointer capable of pointing to a current position within the index buffer;

a start of data pointer capable of indicating a location of the beginning of the data within the object buffer;

a data buffer size capable of specifying a size of the object buffer; and

a data pointer capable of pointing to a current position within the object buffer.

9. In a computer system having main memory and persistent memory, the main memory containing at least one data buffer, the method comprising the steps of:

receiving a request for an object, the request comprising an identifier of the object; and

based on the identifier determining whether a cached copy of the object is stored in a plurality of data structures, the data structures comprising:

a cyclic index buffer capable of storing index entries comprising an index, the index having a beginning;

a cyclic object buffer capable of storing and retrieving objects by logical block number, the object buffer having a beginning; and

a metadata buffer capable of storing information about the index buffer and the object buffer.

10. A method as in claim 9, wherein the object is a web object.

11. A method as in claim 9, wherein determining whether a cached copy of the object is stored in a plurality of data structures further comprises the steps of:

computing a hash value of the identifier; and

accessing a hash table using the hash value.

12. A method as in claim 11, wherein the identifier is a uniform resource locator.

13. A method as in claim 9, further comprising:

determining whether a start of index pointer points to a valid index block; and

scanning the index until a valid index block is found or the index is determined to be empty, if the start of index pointer does not point to a valid index block.

14. Apparatus having main memory and persistent memory, the main memory containing at least one data buffer, the apparatus having an execution unit capable of executing program code, and the apparatus comprising:

a receiver for receiving an object from an origin server; and

an object writer for storing the object in a plurality of data structures, the data structures comprising:

a cyclic index buffer capable of storing index entries comprising an index, the index having a beginning;

a cyclic object buffer capable of storing and retrieving objects by logical block number, the object buffer having a beginning; and

a metadata buffer capable of storing information about the index buffer and the object buffer.

15. The apparatus according to claim 14, wherein the object is a web object.

16. The apparatus according to claim 14, further comprising:

a current disk determiner for determining whether a current disk data buffer is full, the current disk data buffer associated with a current disk; and

program code for setting a current disk indicator to indicate a longest waiting disk if the current disk data buffer is full.

17. The apparatus according to claim 14, wherein the object writer further comprises:

an index block writer for storing index blocks, the index blocks comprising:

a time stamp capable of storing a time associated with an index entry; and

a cell number capable of identifying a cell within the cyclic index buffer.

18. The apparatus according to claim 17, wherein the index block writer further comprises an index element writer for writing index elements, the index elements including:

an object identifier capable of identifying an object;

a hash value corresponding to the object identifier;

a cell number corresponding to a storage location of the object; and

a cell size corresponding to the size of the cell within the index buffer.

19. The apparatus according to claim 18, wherein the object is a web object and the identifier is a uniform resource locator.

20. The apparatus according to claim 14, wherein the cyclic data buffer further comprises:

cells comprising at least one data block.

21. The apparatus according to claim 14, wherein the metadata buffer further comprises:

a timestamp capable of storing a time associated with information stored in the metadata buffer;

a start of index indicator capable of indicating a location of the beginning of the index within the index buffer;  
 an index buffer size capable of specifying a size of the index buffer;  
 an index pointer capable of pointing to a current position within the index buffer;  
 a start of data pointer capable of indicating a location of the beginning of the data within the object buffer;  
 a data buffer size capable of specifying a size of the object buffer; and  
 a data pointer capable of pointing to a current position within the object buffer.

22. A computer system having main memory and persistent memory, the main memory containing at least one data buffer, the computer system having an execution unit capable of executing program code, and the computer system comprising:

a receiver for receiving a request for an object, the request comprising an identifier of the object; and  
 a determiner for determining whether a cached copy of the object is stored in a plurality of data structures, based on the identifier, the data structures comprising:  
 a cyclic index buffer capable of storing index entries comprising an index, the index having a beginning;  
 a cyclic object buffer capable of storing and retrieving objects by logical block number, the object buffer having a beginning; and  
 a metadata buffer capable of storing information about the index buffer and the object buffer.

23. The computer system according to claim 22, wherein the determiner for determining whether a cached copy of the object is stored in a plurality of data structures further comprises:

program code for computing a hash value of the identifier; and  
 program code for accessing a hash table using the hash value.

24. The computer system according to claim 22, further comprising:

program code for determining whether a start of index pointer points to a valid index block; and  
 program code for scanning the index until a valid index block is found or the index is determined to be empty, if the start of index pointer does not point to a valid index block.

25. A computer-readable medium capable of causing a computer system to perform a method, the computer system having main memory and persistent memory, the main memory containing at least one data buffer, and the method comprising the steps of:

receiving an object from an origin server; and  
 storing the object in a plurality of data structures, the data structures comprising:  
 a cyclic index buffer capable of storing index entries comprising an index, the index having a beginning;

a cyclic object buffer capable of storing and retrieving objects by logical block number, the object buffer having a beginning; and

a metadata buffer capable of storing information about the index buffer and the object buffer.

26. A computer-readable medium as in claim 25, wherein the object is a web object.

27. A computer-readable medium as in claim 25, further comprising the steps of:

determining whether a current disk data buffer is full, the current disk data buffer associated with a current disk; and

setting a current disk indicator to indicate a longest waiting disk if the current disk data buffer is full.

28. A computer-readable medium as in claim 25, wherein the step of storing objects further comprises the step of:

storing index blocks comprising:

a time stamp capable of storing a time associated with an index entry; and

a cell number capable of identifying a cell within the cyclic index buffer.

29. A computer-readable medium as in claim 28, wherein the step of storing index blocks further comprises the step of storing index elements including:

an object identifier capable of identifying an object;  
 a hash value corresponding to the object identifier;  
 a cell number corresponding to a storage location of the object; and  
 a cell size corresponding to the size of the cell within the index buffer.

30. A computer-readable medium as in claim 25, wherein the cyclic data buffer further comprises:

cells comprising at least one data block.

31. A computer-readable medium as in claim 25, wherein the metadata buffer further comprises:

a timestamp capable of storing a time associated with information stored in the metadata buffer;

a start of index indicator capable of indicating a location of the beginning of the index within the index buffer;

an index buffer size capable of specifying a size of the index buffer;

an index pointer capable of pointing to a current position within the index buffer;

a start of data pointer capable of indicating a location of the beginning of the data within the object buffer;

a data buffer size capable of specifying a size of the object buffer; and

a data pointer capable of pointing to a current position within the object buffer.

32. A computer-readable medium capable of causing a computer system to perform a method, the computer system having main memory and persistent memory, the main memory containing at least one data buffer, and the method comprising the steps of:

receiving a request for an object, the request comprising an identifier of the object; and

based on the identifier, determining whether a cached copy of the object is stored in a plurality of data structures, the data structures comprising:

a cyclic index buffer capable of storing index entries comprising an index, the index having a beginning;

a cyclic object buffer capable of storing and retrieving objects by logical block number, the object buffer having a beginning; and

a metadata buffer capable of storing information about the index buffer and the object buffer.

33. A computer-readable medium as in claim 32, wherein determining whether a cached copy of the object is stored in a plurality of data structures further comprises the steps of:

computing a hash value of the identifier; and

accessing a hash table using the hash value.

34. A computer-readable medium as in claim 33, wherein the identifier is a uniform resource locator.

35. A computer-readable medium as in claim 32, the method further comprising:

determining whether a start of index pointer points to a valid index block; and

scanning the index until a valid index block is found or the index is determined to be empty, if the start of index pointer does not point to a valid index block.

\* \* \* \* \*



US006098096A

**United States Patent** [19][11] **Patent Number:** **6,098,096****Tsirigotis et al.**[45] **Date of Patent:** **Aug. 1, 2000**

[54] **METHOD AND APPARATUS FOR DYNAMIC CACHE PRELOADING ACROSS A NETWORK**

[75] Inventors: **Panagiotis Tsirigotis**, Mountain View;  
**Sanjay R. Radla**, Fremont, both of Calif.

[73] Assignee: **Sun Microsystems, Inc.**, Palo Alto, Calif.

[21] Appl. No.: **08/762,705**

[22] Filed: **Dec. 9, 1996**

[51] Int. Cl.<sup>7</sup> ..... **G06F 15/167**

[52] U.S. Cl. .... **709/213; 709/203; 709/216; 709/217; 709/229; 707/8; 707/10; 707/205**

[58] Field of Search ..... **395/200.3-200.33, 395/200.41-200.49, 200.57-200.59, 200; 707/8-10, 200, 1-2, 205; 709/202-203, 213-219, 227-229**

[56] **References Cited**

**U.S. PATENT DOCUMENTS**

5,329,619	7/1994	Page et al. ....	709/203
5,452,447	9/1995	Nelson et al. ....	707/205
5,680,573	10/1997	Rubin et al. ....	711/129
5,701,461	12/1997	Dalal et al. ....	707/4
5,721,827	2/1998	Logan et al. ....	709/217
5,727,065	3/1998	Dillon ....	380/49
5,737,536	4/1998	Hermann et al. ....	709/229
5,802,292	9/1998	Mogul ....	709/203
5,940,594	8/1999	Ali et al. ....	709/203

**OTHER PUBLICATIONS**

Reprinted from Web Week, vol. 2, Issue 9, Jul. 8, 1996, Mecklermedia Corp., URL=<http://www.webweek.com/Jul.8,1996/comm/rating.html>.

Azer Bestavros, "Speculative Data Dissemination and Service", Data Engineering, 1996, pp. 180-187.

Zhenig Wang, et al., "Prefetching In World Wide Web", Communications: The Key To Global Prosperity, Globecom, 1996, Global Internet 96 Conference Record, London, Nov. 18-22, 1996, vol. Supp., Nov. 18, 1996, pp. 28-32.

Bill N. Schilit, et al., "TeleWeb: Loosely Connected Access To The World Wide Web", Computer Networks and ISDN Systems, vol. 28, No. 11, May 1996, pp. 1431-1444.

James S. Gwertzman et al., "The Case For Geographical Push-Caching", Proceedings Fifth Workshop On Hot Topics in Operating Systems (HOTOS-V) (Cat. No. 95<sup>th</sup> 8059), Proceedings 5<sup>th</sup> Workshop On Hot Topics In Operating Systems (HOTOS-V), Orcas Island, WA, USA, May 4-5, 1995, pp. 51-55.

*Primary Examiner*—Zarni Maung

*Assistant Examiner*—Bharat Barot

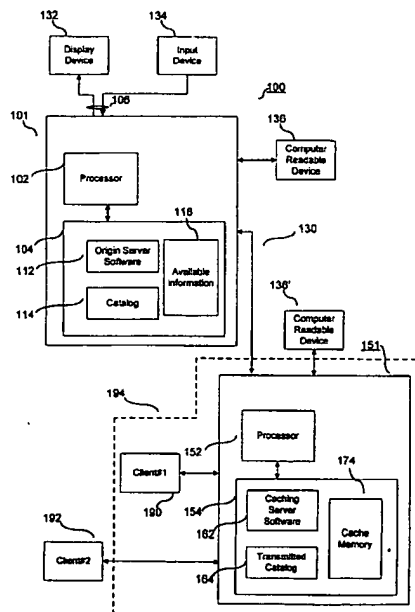
*Attorney, Agent, or Firm*—O'Melveny & Myers LLP

[57]

**ABSTRACT**

A caching server that provides faster access times for independently operating network elements. The caching server initiates information transfer and holds the requested information in its memory, instead of caching information transfer in response to user requests. The caching server preloads information from another server into its memory based on a set of predetermined criteria. Such preloading preferably occurs during low usage time, such as nighttime. The information source determines which information is described in the catalog in accordance with one of a second set of predetermined criteria. For example, the catalog may be organized according to size and the information source will organize the catalog by file size. The caching server preloads as many often-used files as will fit in its cache memory

**20 Claims, 4 Drawing Sheets**



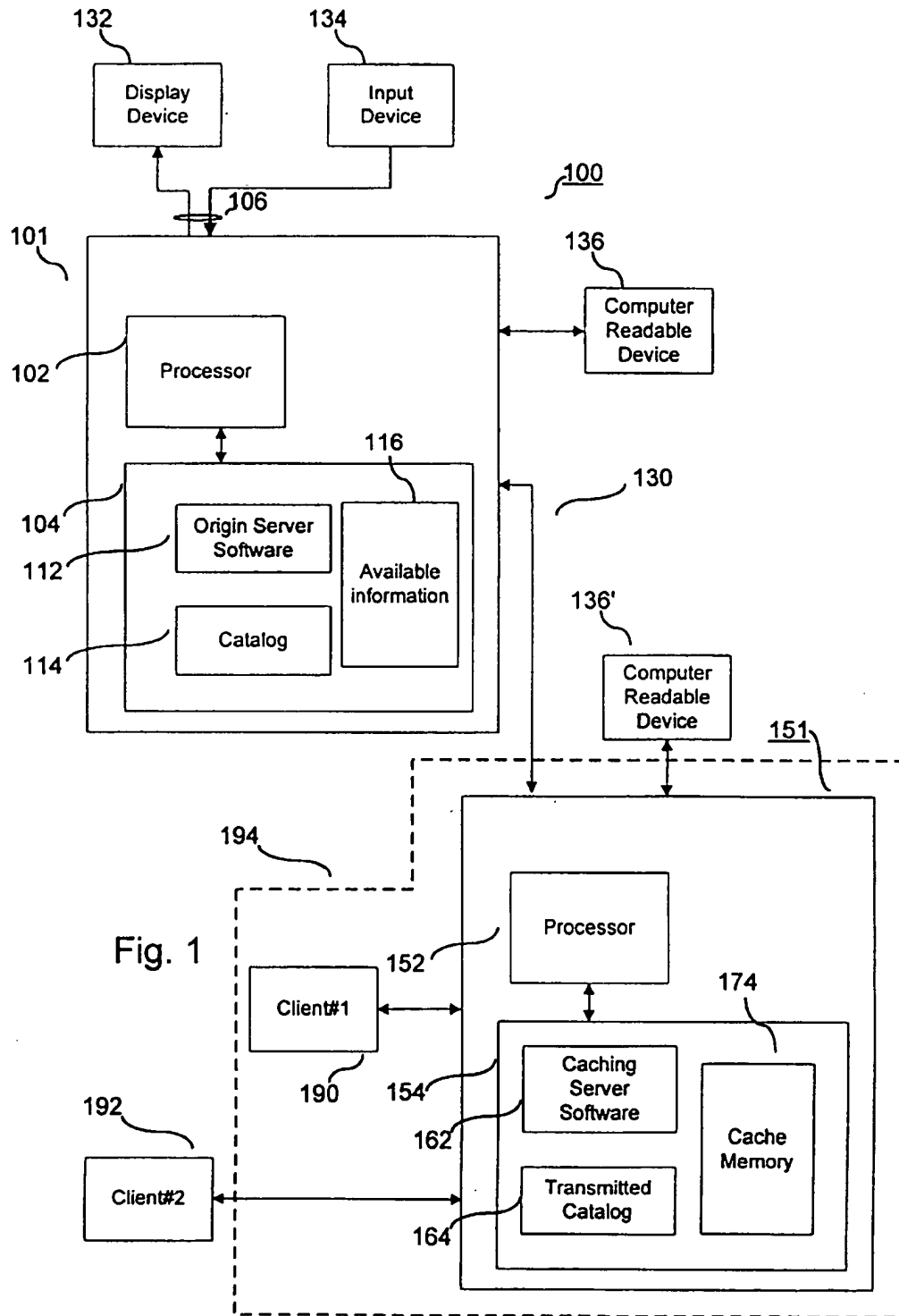
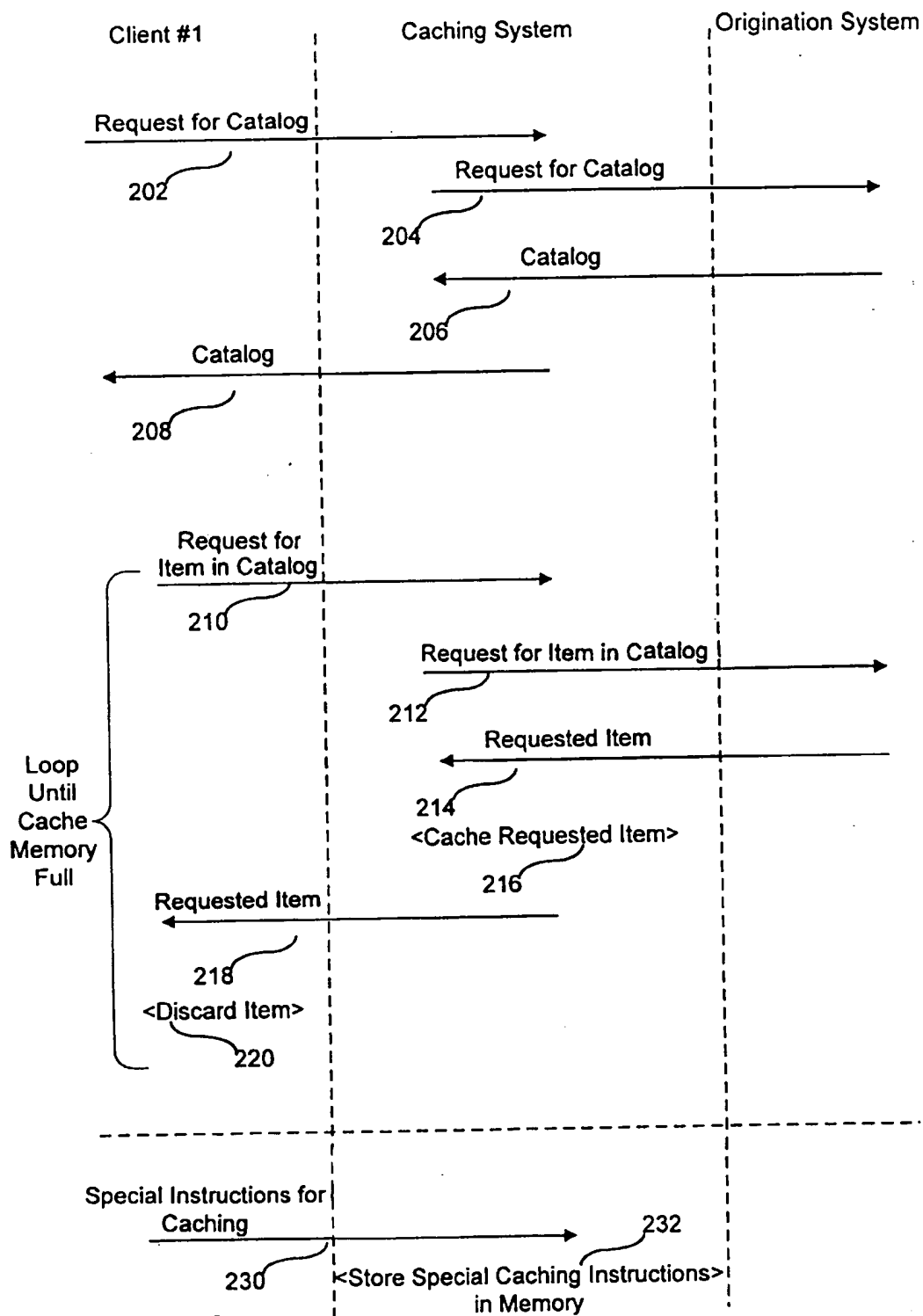


Fig. 1



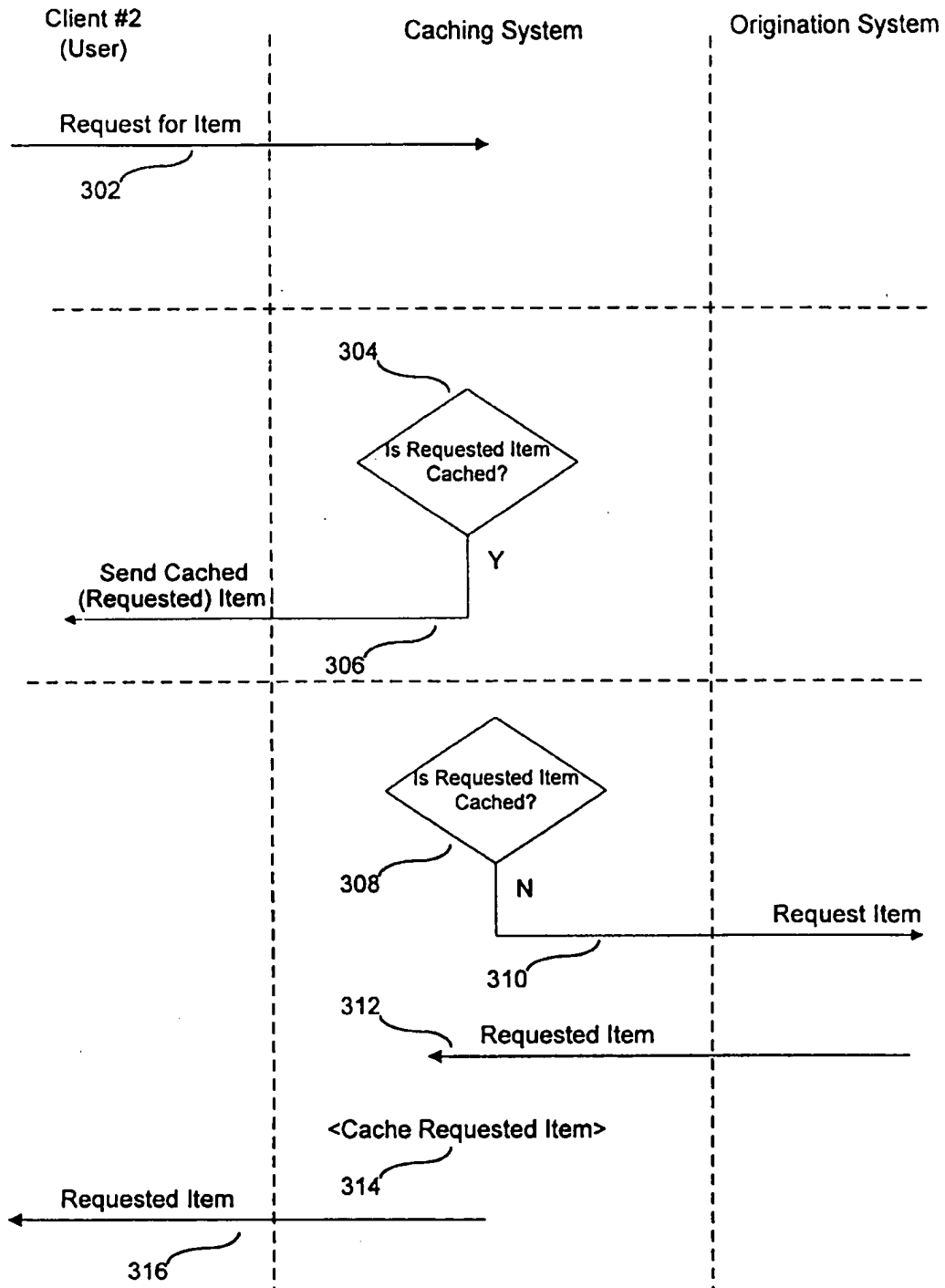


Fig. 3

114	402 URL	404 SIZE

Fig. 4(a)

114'	410 URL	412 SIZE	414 # TIMES DOWNLOADED
			-

Fig. 4(b)

114''	420 URL	422 SIZE	424 DESIRABILITY
			-

Fig. 4(c)

# METHOD AND APPARATUS FOR DYNAMIC CACHE PRELOADING ACROSS A NETWORK

## RELATED APPLICATIONS

The following co-pending patent applications, which were filed on Dec. 9, 1996, are related to the subject application and are herein incorporated by reference:

- 1) U.S. application Ser. No. 08/763,234, entitled "Method and Apparatus for Client-Sensitive Name Resolution Using DNS" of Swee Boon Lim, Sanjay R. Radia, and Thomas Wong, now U.S. Pat. No. 6,014,660.
- 2) U.S. application Ser. No. 08/762,393, entitled "Method and Apparatus for Controlling Access to Services Within a Computers Network" of Thomas Wong, Sanjay R. Radia, Swee Boon Lim, Panagiotis Tsirigotis, and Rob Goodman, now U.S. Pat. No. 5,835,727.
- 3) U.S. application Ser. No. 08/762,402, entitled "Method and Apparatus for Dynamic Packet Filter Assignments" of Sanjay R. Radia, Swee Boon Lim, Panagiotis Tsirigotis, Thomas Wong, and Rob Goodman, now U.S. Pat. No. 5,848,233.
- 4) U.S. application Ser. No. 08/763,289, entitled "Load Balancing and Failover of Network Services" of Swee Boon Lim, Ashish Singhai, and Sanjay R. Radia, now U.S. Pat. No. 5,938,732.
- 5) U.S. application Ser. No. 08/763,068, entitled "Secure DHCP Server" of Swee Boon Lim, Sanjay R. Radia, Thomas Wong, Panagiotis Tsirigotis, and Rob Goodman, now U.S. Pat. No. 5,884,024.
- 6) U.S. application Ser. No. 08/763,212, entitled "A Method to Activate Unregistered Systems in a Distributed Multi-server Network Environment" of Thomas Wong and Sanjay R. Radia, still pending.
- 7) U.S. application Ser. No. 08/762,709, entitled "A Method and Apparatus for Assignment of IP Addresses" of Thomas Wong, Swee Boon Lim, Sanjay R. Radia, Panagiotis Tsirigotis, and Rob Goodman, still pending.
- 8) U.S. application Ser. No. 08/762,933, entitled "A Method for Using DHCP to Override Learned IP Addresses in a Network" of Sanjay R. Radia, Thomas Wong, Swee Boon Lim, Panagiotis Tsirigotis, Rob Goodman, and Mike Patrick, now U.S. Pat. No. 5,922,049.

The following co-pending patent application is related to the subject application and is herein incorporated by reference:

- 9) U.S. application Ser. No. 08/673,951, filed Jul. 1, 1996, entitled "A Name Service for a Redundant Array of Internet Servers" of Swee Boon Lim still pending.

## FIELD OF THE INVENTION

This application relates to networking and, specifically, to a method and apparatus for caching in a network where the elements of the network are operating independently.

## BACKGROUND OF THE INVENTION

Recent years have seen a huge rise in the popularity of network systems, such as the internet. A network is formed of independently operating elements that are interconnected in a way that allows them to transmit information. For example, a first network element can transmit World Wide Web pages to a second network element. The World Wide Web allows a user to use "web browser" software on his computer system to view "web pages" stored on other systems in the network. Similarly, network elements can transfer files to other network elements via File Transfer Protocol (FTP).

The amount of information transmitted over networks such as the internet has grown and will continue to grow in the future. While the overall amount of information being transferred has grown, the amount of information being transferred during a day still varies at different times of the day. A time of day when the most information is being transferred is called a "peak usage time." A time of day when a relatively small amount of information is being transferred is called a "low usage time." For example, a network that allows a consumer to download information to a personal computer using a cable modem might have a peak usage time when people get home from work and a low usage time in the middle of the night when people are asleep. It is desirable that usage of the network be evened out as much as possible. It is also desirable that response time be increased as much as possible.

Moreover, in conventional networks, various elements of the network have different goals. For example, in the above, mentioned consumer network, a centrally located network element may have a goal of sending as much information as possible out over the network. In contrast, a "local" computer may have a goal of giving "its" users the best response time possible.

Local computers often "cache" incoming information. For example, a web browser may save in its memory a copy of the ten most recent pages viewed by the user. That way, if the user wants to view any of the cached web pages, the browser software does not have to send a request for the web page out over the network. Instead, the browser can merely get the cached web page from its memory. The browser can access its memory much faster than it can request a web page and receive it over the network. Thus, caching saves the time otherwise required to fetch the web page from over the network. Local computers often cache the most recently accessed information so that the information will be available should the user request it again. A problem associated with conventional caching is that the user must wait for information to be downloaded into the local computer when the user makes a first request for the information. (Caching is performed in response to a user request for information). A second problem is that the information to be cached is often requested and downloaded during peak usage times.

## SUMMARY OF THE INVENTION

The present invention overcomes the problems and disadvantages of the prior art by implementing a caching server that provides faster access times for independently operating network elements. Such independently operating network elements are said to have "loosely coupled administrative domains." In a preferred embodiment of the present invention, the caching server initiates information transfer and holds the requested information in its memory, instead of caching information transfer in response to user requests.

In a preferred embodiment of the present invention, the caching server preloads information from another server into its memory based on a set of predetermined criteria. For example, the caching server may download as many of the largest files as will fit in its cache memory. Such preloading preferably occurs during low usage time. Specifically, the caching server requests a "catalog" of available information from an information source, such as another server. The information source sends the catalog over the network. The caching server then decides which information it wants to preload and requests that information from the information source. Once the information is received, the caching server holds the information in a cache memory so that the information will be available when requested by a user.

Cache preloading has two main benefits: 1) it eliminates the latency time associated with accessing information requested by a user for the first time and 2) the information fetch between the information source and the caching server can take place during off-peak hours, thereby reducing the network load at peak times.

In a preferred embodiment of the present invention, the information source determines which information is described in the catalog in accordance with one of a set of predetermined criteria. For example, the catalog may be organized according to size and the information source will organize the catalog by file size. As a second example, the catalog may be organized according to the files most-often downloaded by users. In this case, the caching server will preload as many often-used files as will fit in its cache memory. As a third example, the catalog may be organized according to which files advertisers have paid to place in the catalog. In this case, the caching server will preload as many of these files as will fit in its cache memory. In accordance with the purpose of the invention, as embodied and broadly described herein, the invention is a method for quickly accessing information in a network that includes an origin system connected to a caching server, the method comprising the steps, performed by the caching server of: requesting a catalog from the origin system that lists documents available from the origin system; receiving the catalog from the origin system; requesting a document described in the catalog, the document requested according to a predetermined criterion of the caching server; receiving the document from the origin system; and storing the document in a cache memory of the caching server.

In further accordance with the purpose of this invention, as embodied and broadly described herein, the invention is a caching server having a cache memory, the caching server being part of a network that includes an origin system connected to the caching server, the caching server comprising: a first portion configured to request a catalog from the origin system that lists documents available from the origin system; a second portion configured to receive the catalog from the origin system; a third portion configured to request a document described in the catalog, the document requested according to a predetermined criterion of the caching server; a fourth portion configured to receive the document from the origin system; and a fifth portion configured to store the document in the cache memory.

Advantages of the invention will be set forth in part in the description which follows and in part will be obvious from the description or may be learned by practice of the invention. The objects and advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims and equivalents.

#### BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate several embodiments of the invention and, together with the description, serve to explain the principles of the invention.

FIG. 1 is a block diagram of a data processing system in accordance with a preferred embodiment of the present invention.

FIG. 2 is a diagram showing steps of a method for preloading a caching server.

FIG. 3 is a diagram showing steps of a method for fulfilling a user's request for information.

FIGS. 4a through 4c show examples of possible catalog formats.

#### DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

FIG. 1 is a block diagram of a computer network 100 in accordance with a preferred embodiment of the present invention. Computer network 100 includes an origin computer system 101 and a caching computer system 151. Origin system 101 and caching system 151 are connected via connection 130. The computer system 100 can be any type of network, such as the internet, a LAN, a WAN, an intranet, etc., that allows systems 101 and 151 to communicate with each other. Origin system 101 and caching system 151 operate independently and communicate using any known protocol, such as Hyper Text Transfer Protocol (http). Although only two systems 101 and 151 are shown, it should be understood that additional systems can be included in computer network 100 without departing from the spirit and scope of the present invention.

Origin computer system 101 includes a processor 102 and a memory 104. Memory 104 includes origin server software 112, available information 116 and a catalog 114. Catalog 114 contains, for example, a description of information available from origin system 101. System 101 preferably connects to a display device 132, a computer readable device 136 and an input device 134, which can be any of a wide range of varying I/O devices, such as disk drives, keyboards, modems, network adapters, printers, and displays. Respective computer readable devices 136, such as a disk drive or CD ROM drive, are shown connected to origin system 101 and caching system 151. Origin server software 112 and Caching server software 162 are preferably loaded into memory via devices 136.

Caching computer system 151 includes a processor 152 and a memory 154. Memory 154 includes caching server software 162 and a catalog 164 transmitted from origin system 101. FIG. 1 also shows a cache memory 174. Cache memory 174 may be a part of memory 104 (as shown) or may be a separate memory. Cache memory 174 has a predetermined maximum size. In the described embodiment, caching system 151 is connected to client#1 190 and client#2 192. In the described embodiment, client#1 190 and caching system 151 jointly form a caching server 194. In other embodiments, client#1 may be located in either the same or a different computer as caching system 151. In still other preferred embodiments, caching system 151 performs the functions of client#1 190. Client#2 may be located in either the same or a different computer as caching system 151.

The present invention allows preloading of caching system 151 from origin system 101, where the priorities of the entities administering the systems 101 and 151 are different. For example, the origin system 101 will want to "push out" as much information as possible, while the caching system 151 is willing only to cache a certain amount of information. Origin system 101 may be, for example, a large, centrally located server that stores a large number of http documents, such as web pages and files. Caching system 151 may be, for example, a regional server accessible by clients in a geographical region. A typical network may include multiple origin systems 101 and/or multiple caching systems 151.

FIG. 2 is a diagram showing steps of a method for preloading caching system 151 in accordance with a pre-

ferred embodiment of the present invention. The figure is divided into three areas, representing, respectively, Client#1 190, caching system 151 and origin system 101. In the described embodiment, the client and the servers communicate with one another via the http protocol, although other suitable protocols could be used. Processor 152 executes caching server software 162 to perform the steps of the middle column. Processor 102 executes origin server software 112 to perform the steps of the right-most column. Client#1 190 and client#2 192 preferably contain a processor executing software performing the illustrated steps. As discussed above, FIG. 1 shows client#1 190 as a separate process, which may or may not run on a separate machine. In an alternate embodiment, the functions of client#1 190 are performed by caching server software 162.

In step 202, client#1 190 initiates a request for a catalog to caching system 151. In the described embodiment, client#1 190 sends such a request at a predetermined time, such as in the middle of the night. It is desirable that the request of step 202 be performed at a time when the network is not busy. Such timing helps to reduce network load at peak times. In step 204, caching system 151 sends a request for a catalog to origin system 101. In step 206, origin system 101 sends the catalog 114 to caching system 151. In step 208, caching system 151 sends the catalog to client#1 190.

Steps 210 through 220 are performed multiple times until client#1 190 determines that cache memory 174 is full (or that some maximum amount of cache space is filled). Alternately, client#1 190 may send a list of information to preload to caching system 151 and caching system 151 will load as many of the items on the list as will fit in cache memory 174. In step 210, client#1 190 requests an item in the catalog from caching system 151. In step 212, caching system 151 sends the request to origin system 101, which in step 214 returns the requested item. In step 216, caching system 151 caches the returned item in cache memory 174. FIG. 2 shows in step 218 that the item is then forwarded to client#1 190, which in step 220 discards it, but may keep information about the item, such as the size. Alternately, caching system 151 may simply cache the item without sending it to client#1 190. Thus, at the end of step 216, information has been preloaded into caching system 151 and is awaiting user requests for the information.

FIG. 2 also shows a step 230 in which client#1 190 sends special instructions to caching system 151. These special instructions may, for example, specify that certain information in cache memory 174 should not be discarded from cache memory for a predetermined amount of time. As another example, the special instruction may instruct caching system 151 to immediately discard certain information in cache memory 174. The information may be specified, for example, by its URL (Uniform Resource Locator). Alternately, the special instruction may instruct the caching system 151 to delete (or save) information in a last-in, first-out order or in some other appropriate order. In step 232 caching system 151 stores special caching instructions in memory 154.

FIG. 3 is a diagram showing steps of a method for fulfilling a user's request for information. The steps of FIG. 3 are performed after the caching system 151 has preloaded some information into cache memory 174, as shown in FIG. 2. In the example, the user is logged onto client#2 192. In step 302, the user requests information such as a file or a web page by URL. If, in step 304, the requested information has been preloaded and exists in cache memory 174, the information is copied from cache memory 174 and in step 306 sent to client#2 192. If, in step 308, the information was not

preloaded, then, in step 310, caching system 151 requests the information from origin system 101, which in step 312 returns the requested information. In step 314, caching system 151 stores the information in cache memory 174 and, in step 316, sends the information to the requesting client. Caching system 151 determines whether certain information is present in cache memory 174 via any of a variety of known cache memory techniques.

Note that step 314 may involve purging (discarding) items from cache 174 to make room for new items. If caching system 151 has received special instructions not to discard certain cached information, then in step 314, that information will not be discarded from cache, even if it would normally be next in line to be discarded.

FIGS. 4a through 4c show respective examples of formats of catalog 114. Each example format includes a section or row in the catalog for each file or URL that is available to be downloaded from the origin system. These formats are provided by way of example only and are not meant to limit the scope of the present invention. Catalog 114 is preferably an http document of type "text/plain." FIG. 4a shows a format in which catalog 114 includes two fields: a Uniform Resource Locator (URL) field 402 and a size field 404. The catalog may optionally be organized by increasing or decreasing size.

FIG. 4b shows a format in which catalog 114 includes three fields: a URL field 410, a size field 412, and a "# of times downloaded" field 414 (also called a "popularity" field). Alternately, the popularity field 414 could be omitted and the catalog could simply be sorted by ascending or descending popularity value.

FIG. 4c shows a format in which the catalog includes three fields: a URL field 420, a size field 422, and a "the desirability field 424." Alternately, the desirability field 412 could be omitted and the catalog could simply be sorted by ascending or descending desirability. In this example, desirability indicates a criteria defined in origin system 101. For example, if an advertiser had paid a premium to have his information placed into the catalog, the information would have a high desirability value in the catalog. Other examples of catalogs might have the URLs sorted according to their PICS rating, or any similar rating which rates the content of files.

Caching system 151 reviews the contents of catalog 114 and determines which available information described in catalog 114 it should preload. This determination is made in accordance with a number of criteria stored in the memory of caching system 151. A simple criteria, for example, is to simply preload files in the catalog by size, largest first, until cache memory 174 is full (or a predetermined cache size limit has been reached). Another possible criteria might be that caching system 151 preloads according to both the size of the file, the available cache space, and the "popularity" or "desirability" of the file. Another possible criteria might be that the caching system 151 preloads in accordance with one of the criteria described above, but does not, under any circumstances, download information on a "do not download" list stored in its memory. For example, certain caching systems 151 might never want to download adult material or material from a predetermined information source. As can be seen from the above examples, the criteria used to determine what information to download is affected somewhat by what information is available in catalog 114. A large number of catalog formats and caching system criteria are compatible with the present invention.

In some embodiments, origin system 101 gives "hints" to the caching system/client as to when it should next request

7

a catalog. These hints may be sent as part of the catalog or may be sent in a separate administrative message. For example, if the catalog is normally updated every twenty-four hours, origin system 101 might suggest that the catalog be downloaded by the caching system every twenty-four 5 hours. If, however, some important event is occurring, origin system 101 might suggest that the catalog be downloaded more frequently because it is being updated more frequently by origin system 101.

In a preferred embodiment, if origin system 101 is 10 accessed by multiple caching servers, a different catalog can be sent to each caching server 194. The catalog is dynamically generated based on, for example, the access pattern generated by the particular caching server as observed by the origin system. Alternately, catalogs are determined by policy 15 decisions at the origin system 101. For example, the catalogs sent to a certain caching server may not contain adult material.

In summary, the present invention allows a caching server preferably including a client and a caching system, to initiate 20 a request for a catalog of available http documents. This request is made at night or during a slow time on the network. After the catalog is received, the caching server decides which of the documents in the catalog to prelo 25 preloaded documents should not be purged from its cache for a predetermined time.

Other embodiments will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope of the invention being indicated by the following claims and equivalents.

What is claimed is:

1. A method for quickly accessing information in a 35 network that includes an origin system and a user computer, each connected to a caching server, the method comprising the steps, performed by the caching server of:

requesting a catalog from the origin system that lists 40 documents available to be downloaded from the origin system, the documents selected by the origin system according to a first predetermined criterion of the caching server;

receiving the catalog from the origin system;

selecting a document from the catalog according to a 45 second predetermined criterion of the caching server:

requesting the selected document described in the catalog from the origin system;

receiving the selected document from the origin system; 50 and

storing the selected document in a cache memory of the caching server, thereby making the selected document available to the user computer through the caching server before a request for the selected document is 55 received from the user computer.

2. The method of claim 1, wherein the user computer is connected to the caching server through the network, the method further including the steps performed by the caching server of:

receiving, by the caching server, a request for the selected 60 document from the user computer; and

sending, from the caching server over the network, the selected document stored in the cache memory to the user computer.

3. The method of claim 2, further including the steps 65 performed by the caching server of:

8

determining a portion of the cache memory to be used for storing preloaded documents; and

repeating the steps of selecting a document, requesting the selected document, receiving the selected document and storing the selected document until the determined portion of the cache memory is full of preloaded documents that meet the second predetermined criterion, or until no further documents in the catalog meet the second predetermined criterion.

4. The method of claim 3, further including the steps of: receiving from the user computer a request for a new document that has not been preloaded into the cache memory of the caching server;

requesting the new document from the origin system;

receiving via the network the new document from the origin system; and

sending via the network the new document to the user computer.

5. The method of claim 4, further including the steps of: storing the new document in the cache memory;

wherein non-preloaded documents stored in the cache memory are purged to free cache memory space when the cache memory is full to make storage space available for the new document.

6. The method of claim 5, further including the step, between the steps of receiving the catalog from the origin system and selecting a document, of purging preloaded documents stored in the determined portion of the cache memory to make storage space available for the selected document.

7. The method of claim 1, wherein the step of requesting a catalog from the origin system performed during a low usage time of the network.

8. The method of claim 1, wherein the caching server includes a client and a caching system and wherein the step of requesting a catalog from the origin system includes the steps of:

sending the request for the catalog from the client to the caching system; and

sending the request for the catalog from the caching system to the origin system.

9. The method of claim 1, wherein the caching server includes a client and a caching system and wherein the step of requesting a document described in the catalog includes the steps of:

sending the request for selected document from the client to the caching system; and

sending the request for selected document from the caching system to the origin system.

10. The method of claim 1, wherein the second predetermined criterion used by the caching server to determine whether to request a document from the catalog is based on the size of the document.

11. The method of claim 1, wherein the second predetermined criterion used by the caching server to determine whether to request a document from the catalog is based on a rating associated with the document and stored in the catalog.

12. The method of claim 1, wherein the second predetermined criterion used by the caching server to determine whether to request a document from the catalog is based on a list of non-acceptable document names.

13. The method of claim 1, wherein the network includes a second caching server and the first and second caching servers receive different catalogs from the origin system.

14. The method of claim 1, wherein the network includes a second caching server and the first and the second caching servers request different documents from the origin system.

15. The method of claim 1, wherein the origin system and the caching server use http protocol to communicate.

16. The method of claim 1, wherein the caching server includes a client and a caching system and further including the steps of:

sending, by the client to the caching system, an instruction to retain the document in the cache memory for a specified amount of time; and

retaining the document in the cache memory, by the caching system, for the specified amount of time.

17. The method of claim 1, wherein the caching server includes a client and a caching system and further including the steps of:

sending, by the client to the caching system, an instruction to purge the document from the cache memory; and

purging the document from the cache memory, by the caching system.

18. A caching server having a cache memory, the caching server being part of a network that includes an origin system and at least one user computer connected to the caching server through the network, the caching server comprising:

a first portion configured to request a catalog from the origin system that lists documents available to be downloaded from the origin system, the documents selected by the origin system according to a first predetermined criterion of the caching server;

a second portion configured to receive the catalog from the origin system;

a third portion configured to select a document from the catalog according to a second predetermined criterion of the caching server;

a fourth portion configured to request the selected document described in the catalog from the origin system;

a fifth portion configured to receive the selected document from the origin system;

a sixth portion configured to store the selected document in the cache memory;

a seventh portion configured to receive a request for the selected document from the at least one user computer; and

a eighth portion configured to send the requested document stored in the cache memory to the at least one user computer.

19. The caching server of claim 18, wherein the caching server includes a client and a caching system and wherein the first portion includes:

a ninth portion configured to send the request for the catalog from the client to the caching system; and

a tenth portion configured to send the request for the catalog from the caching system to the origin system.

20. A computer program product comprising:

a computer usable medium having computer readable code embodied therein for causing access to information stored on a network having a origin system, a caching server, and at least one user computer, the computer program product comprising:

computer readable program code devices configured to cause a computer to effect requesting a catalog from the origin system that lists documents available to be downloaded from the origin system, the documents selected according to a first predetermined criterion of the caching server;

computer readable program code devices configured to cause a computer to effect receiving the catalog from the origin system;

computer readable program code devices configured to cause a computer to effect selecting a document from the catalog according to a second predetermined criterion of the caching server;

computer readable program code devices configured to cause a computer to effect requesting the selected document described in the catalog from the origin system;

computer readable program code devices configured to cause a computer to effect receiving the selected document from the origin system;

computer readable program code devices configured to cause a computer to effect storing the selected document in the cache memory;

computer readable program code devices configured to cause a computer to effect receiving a request for the selected document from the at least one user computer; and

computer readable program code devices configured to cause a computer to effect sending the requested document stored in the cache memory to the user computer.

\* \* \* \* \*